Bureau canadien des brevets

Certification

La présente atteste que les documents ci-joints, dont la liste figure ci-dessous, sont des copies authentiques des documents déposés au Bureau des brevets.

Canadian Patent Office

Certification

This is to certify that the documents attached hereto and identified below are true copies of the documents on file in the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial No: **2,414,053**, on December 9, 2002, by **COREL CORPORATION,** assignee of Gordon Bowman and Peter Barrett, for "System and Method for Manipulating a Document Object Model".

Sylvie Grégoire

Agent certificateur/Certifying Officer

July 16, 2003

Date

Abstract

A system for manipulating a document object model is provided. The system comprises a collection of document object model behavior elements, a collection of scripts for performing actions associated with the set of behavior elements, each script associated with a behavior element, and an initialization function for directing the processing of one or more behavior elements in a document object model. Each behavior element comprises a name following a predetermined naming convention, an event attribute for associating the behavior element to an event, and other attributes for describing features of the behavior element.

System and Method for Manipulating a Document Object Model

FIELD OF THE INVENTION

The invention relates to markup languages. In particular, the invention relates to a system and method for manipulating a document object model.

BACKGROUND OF THE INVENTION

Web designers use markup languages to create and modify web sites. A document object model (DOM) is created by a viewer when accessing an extensible markup language (XML) based file. It is often desirable to manipulate the DOM.

Figure 1 shows a typical web display environment 10 for displaying web pages and web applications. A web display environment 10 comprises a browser 11, a viewer 13, a script interpreter 14, and a DOM 15. The browser 11 is the host application, which understands and visually renders hypertext markup language (HTML) and/or extensible hypertext markup language (XHTML). Examples of browsers include Netscape (TM) and Internet Explorer (TM). The browser 11 includes a window which is displayed on the display apparatus, such as a monitor, of an end user computer system. The browser 11 typically employs a plug-in architecture, in which third party software (known as the plug-in or viewer 13) can be associated with any file format that is not already natively supported by the browser 11 and is allowed to render that file within the host browser's 11 window. One type of file that the browser 11 may be asked to open is a Scalable Vector Graphic (SVG) file having a ".svg" extension. The browser 11 does not natively support the SVG markup language (which is an XML language) and so passes the SVG file to the SVG viewer 13, which has associated itself to the SVG file format, via the rules of the plug-in architecture of the browser 11.

The viewer 13 comprises software code for parsing the SVG markup, creating a DOM, rendering that DOM to the browser's window, listening for events and dispatching them to their assigned handler script functions, and interpreting/executing those script functions. An example of a viewer 13 is the Corel (TM) SVG Viewer. The viewer 13 uses the SVG file received from the browser 11 to create a DOM 15. The DOM is a hierarchical tree structure of objects in memory, representing the hierarchical XML markup in the XML text file. The DOM also contains methods (also known as functions or application programming interfaces (API's)) that allow it to be queried or modified. The viewer 13 may also have access to a script interpreter/engine, which can execute script code 14 created by a programmer for the purpose of making the document non-static (e.g., animation) and/or interactive with the user (e.g., the user can create events with the mouse or keyboard, which cause something to happen) via manipulation of the DOM.

One way to manipulate a DOM is to use scripting. However, many web designers do not have the programming skills required for DOM manipulation via scripting. Thus, programmers are needed to create the scripts for the designer. Programmers can be costly, plus it can take a long time to develop stable, fast code. Thus, it is desirable to have a system or method of manipulating a DOM that a designer with minimal programming knowledge may operate, and which could also aid even an experienced programmer to rapidly develop a web application.

One way of assisting designers and developers is to have pre-canned scripts for the most commonly required functionality. However, supporting the insertion of pre-canned scripts 14 via an integrated development environment (IDE) is both complicated and limiting. For example, the Microsoft (TM) Visual Studio IDE can create auto-generated code for its Microsoft Foundation Classes (MFC) (which abstract the programmer from the core Win32 API's), making it easier and quicker to program Windows applications. However, limits must be imposed on the user. User-modification of the auto-generated code is discouraged, because it makes it difficult to regenerate the code from the project file, or to automatically modify the pre-generated code as a result of new user-defined parameters to the abstractions.

Software exists that allows one to map input XML markup to output markup. It is difficult to generate data-driven script via the Extensible Stylesheet Language Transformation (XSLT), the most commonly used markup language for transforming XML markup to a different form of markup.

Scripts 14 are interpreted, and thus provide inherently slower performance than what can be achieved with natively implemented code. Scripts 14 can only manipulate the DOM 15 via the DOM application programming interfaces (API's) that are exposed to the programmer, which may be abstractions on top of the real object model used by the viewer 13, which can only be accessed by native code. Scripts 14 add to the amount of data needed to be transferred. This volume of data is especially a problem for wireless devices with low bandwidth. Finally, scripts 14 are only as powerful as the DOM API's that the viewer 13 supports. Currently, not all viewers 13 support the entire spectrum of DOM API's.

The Synchronized Multimedia Integration Language (SMIL) has a <set> element that allows one to set the attributes of an element within the current document to a specific, known value. However, this <set> element does not actually call the DOM's setAttribute() method. Thus it does not fire a mutation event (informing any listener function that the attribute has changed). Also, the <set> element does not actually result in changes to the "core DOM", which represents the document, but rather it results in changes to the "animated DOM", which stores animated values for each attribute which may be animated. Furthermore, there is no way to monitor changes to the animated DOM. Also, the <set> element cannot affect other elements in other documents, and cannot be easily linked to events and cannot reference attributes in other elements. So, while SMIL can result in visual animation, it cannot be used for full DOM 15 manipulation. Only script can do that by accessing the DOM methods.

SUMMARY OF THE INVENTION

It is an object of the invention to provide a novel system and method of manipulating a document object model that obviates or mitigates at least one of the problems described above.

In an aspect of the present invention, there is provided a system for manipulating a document object model. The system comprises a collection of document object model behavior elements, a collection of scripts for performing actions associated with the set of behavior elements, each script associated with a behavior element, and an initialization function for directing the processing of one or more behavior elements in a document object model. Each behavior element comprises a name following a predetermined naming convention, an event attribute for associating the behavior element to an event, and other attributes for describing features of the behavior element.

In another aspect of the present invention, there is provided a system for manipulating a document object model. The system comprises a collection of scripts for performing actions associated with markup behavior elements, and an initialization function for directing the processing of one or more behavior elements in a document object model. Each script associated with a behavior element.

In another aspect of the present invention, there is provided a method of manipulating a document object model. The method comprises the steps of searching for a designated element in a document object model, and calling a script associated with the designated element.

In another aspect of the present invention, there is provided a method of manipulating a document object model. The method comprises the steps of adding an event listener to an element having a designated element as a child in the document object model, receiving an event which is equal to an event attribute setting in the designated element, and calling a script associated with the designated element.

In another aspect of the present invention, there is provided a method of creating an element for manipulating a document object model. The method comprises the steps of categorizing low level actions into behavior groupings, determining common attributes of a behavior grouping, and creating a behavior element having the common attributes of the behavior grouping.

In another aspect of the present invention, there is provided computer readable media storing the instructions and/or statements for use in the execution in a computer of a method of manipulating a document object model. The method comprises the steps of searching for a designated element in a document object model, and calling a script associated with the designated element.

In another aspect of the present invention, there is provided electronic signals for use in the execution in a computer of a method of manipulating a document object model. The method comprises the steps of searching for a designated element in a document object model, and calling a script associated with the designated element.

In another aspect of the present invention, there is provided a computer program product for use in the execution in a computer of a method of manipulating a document object model. The computer program product comprises a collection of scripts for performing actions associated with markup behavior elements, and an initialization function for directing the processing of one or more behavior elements in a document object model. Each script associated with a behavior element.

BRIEF DESCRIPTIONS OF THE DRAWINGS

Figure 1 shows a typical web display environment.

Figure 2 shows a document object model manipulation system, in accordance with an embodiment of the present invention.

Figure 3 is a pictorial representation of a browser window with a circle in the top of the browser window.

Figure 4 is a node tree representation of an example of a document object model, in accordance with an embodiment of the present invention.

Figure 5 is a pictorial representation of a browser window with two circles in the browser window.

Figure 6 is a flowchart of an example of a method of manipulating a document object model at load time, in accordance with an embodiment of the present invention.

Figure 7 is a flowchart of an example of a method of manipulating a document object model in response to an event, in accordance with an embodiment of the present invention.

Figure 8 is a flowchart of another example of a method of manipulating a document object model, in accordance with an embodiment of the present invention.

Figure 9 is a flowchart of an example of a method of creating an element for manipulating a document object model, in accordance with an embodiment of the present invention.

Figure 10 is a flowchart of another example of a method of creating an element for manipulating a document object model, in accordance with an embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

Figure 2 shows a system for manipulating a Document Object Model (DOM) 20 in accordance with an embodiment of the present invention. The DOM manipulation system

20 comprises a collection (or set) of DOM behavior elements 22, a collection (or set) of scripts 26, and an initialization file 21. Each behavior element 22 comprises a name 23, an 'event' attribute 24, and other attributes 25. The name 23 of the behavior element 22 follows a predetermined naming convention. These behavior elements are not currently part of any official extensible markup language (XML) standard, and are follow a naming convention of adding the namespace "dsvg:" as a prefix to the element name, so that the viewer's 13 XML parser allows it to be part of the DOM. The 'event' attribute 24 associates the behavior element 22 to an event, which will trigger its execution (i.e., make it perform whatever action it is supposed to perform). An example of an event is 'onclick', which is generated by clicking the mouse while the mouse pointer is overtop of a displayed element. Thus, when a user clicks on a displayed item associated with an object in the DOM, the behavior element 22 is executed. Until such time as the event associated with the event specified by the behavior element's 'event' attribute occurs, the behavior element remains dormant. The other attributes 25 describe the details of what the behavior element 22 is supposed to do. For example, other attributes 25 may include the identification (ID) of an object in the DOM, or in the case of the <dsvg:setAttribute> element, the 'name' attribute, which specifies the name of the attribute to be modified (e.g., 'width'), and the 'value; attribute, which specifies the value to which the attribute is to be set (e.g., 100).

The initialization function 21 contains instructions for traversing each node in the DOM immediately after the DOM has been created, searching for the behavior elements by searching for any element whose name is prefixed with the desired namespace (e.g., "dsvg:"), and calling the script 26 that is associated with each particular behavior element, whose name follows the predetermined naming convention. The scripts 26 perform functionality associated with the corresponding elements 22. Preferably, there is a one-to-one relationship between a behavior element 22 and its associated script 26. Preferably, a script 26 is created that can detect which viewer 13 it is being run on, and if the application programming interfaces (API's) are not available, work around the deficiency. Preferably, the initialization function 21 and the scripts 26 are stored in a predetermined format either in the document text file or in a separate text file on a file system or webserver.

An alternative DOM manipulation system comprises the initialization file 21 and the scripts 26 of the DOM manipulation system 20. The collections of behavior elements are provided independently from the alternative DOM manipulation system as markup syntax for a designer (or developer, or any user) to use when modifying an XML file, such as an SVG document.

The following is an example of the syntax of a behavior element 22:

```
<dsvg:copyElement
      id="string"
      event="string"
      newElementID="string"
      {source="xpath" | sourceFrameID="string"
sourceObjectID="string" sourceDocID="string"
sourceElementID="string"}
      {target="xpath" | targetFrameID="string"
targetObjectID="string" targetDocID="string"
targetElementID="string"}
      insertAs="{child | parent | sibling |
replacement}"
      offset="signed integer"
      from="{top | bottom}"
      copyChildren="{true | false}"
      copyEvents="{true | false}"
      copyAttributes="{all | none |
attr1;attr2;...attrN}"
      preserveTargetChildren="{true | false}"
      preserveTargetEvents="{true | false}"
      preserveTargetAttributes="{all | none |
attr1;attr2;...attrN}"
/>
```

The <copyElement> element creates a copy of an existing element and inserts the copied element into the DOM at a desired location. The <dsvg:copyElement> element contains the name 23:

```
dsvg:copyElement
```

which follows the naming convention of adding "dsvg:" as a prefix to the element name. As will be described below, this naming convention will assist the system 20 to search for behavior elements, i.e., all elements having this prefix in their name 23. The <dsvg:copyElement> element also contains the event attribute 24:

```
event="string"
```

The 'event' attribute 24 is set to an event which will trigger the behavior element 22. The attribute of the event attribute is the name of the event which is entered as a string. Examples of an event include "onload", "onclick", and "onmouseover". The event "onload" instructs the behavior element 22 to activate (i.e., to be processed) whenever the SVG element to which the behavior element is associated (via the <dsvg:listener> element or by nature of bing a child of the SVG element) has received the 'onload' event, which occurs after the entire DOM has been created and all the scripts loaded into memory. The 'onclick' event instructs the behavior element 22 to activate whenever the SVG element to which the behavior element 22 is associated receives the 'onclick' event, which could be caused by the user clicking the mouse button while the mouse cursor is overtop of the SVG element. The 'onmouseover' event instructs the behavior element 22 to activate whenever the SVG element to which the behavior element 22 is associated receives the 'onmouseover' event, which could be caused by the user positioning the mouse overtop of the SVG element. 3The strings given in the above examples may be modified as desired. Other events may be associated with the event attribute.

In this example, the <dsvg:copyElement> element comprises other attributes 25. The 'id' attribute allows this behavior element 22 to be referenced later. The 'newElementID' attribute specifies the value of the 'id' attribute of the newly created element. The 'source' attribute is the XPath (a language for addressing parts of an XML document) pointing to the element to be copied. If the 'source' attribute is provided, the 'sourceFrameID', 'sourceObjectID', 'sourceDocID' and 'sourceElementID' attributes are ignored. The 'sourceFrameID' attribute specifies the 'id' attribute of the frame (e.g., a hypertext markup language (HTML) <frame> element) in which to find the element to be copied. If the 'sourceFrameID' attribute is not provided, the current frame is assumed. The 'sourceObjectID' attribute specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element to be copied. If the 'sourceObjectID' attribute is not provided, the current object is assumed. The 'sourceDocID' attribute specifies the "id" attribute of the document (e.g., a scalable vector graphics (SVG) or extensible hypertext markup language (XHTML) document) in which to find the element to be copied. If the 'sourceDocID' attribute is not provided, the current document is assumed. The 'sourceElementID' attribute specifies the 'id' attribute of the element to be copied.

The 'target' attribute is the XPath pointing to the element at which to insert the new element. If the 'target' attribute is provided, the 'targetFrameID', 'targetObjectID', 'targetDocID' and 'targetElementID' attributes are ignored. The 'targetFrameID' attribute specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to place the new element. If the 'targetFrameID' attribute is not provided, the current frame is assumed. The 'targetObjectID' attribute specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the new element. If the 'targetObjectID' attribute is not provided, the current object is assumed. The 'targetDocID' attribute specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to place the new element. If the 'targetDocID' attribute is not provided, the current document is assumed. The 'targetElementID' attribute specifies the 'id' attribute of the element at which to insert the new element.

The copied element may be inserted at any position in the DOM relative to the target element. The 'insertAs' attribute specifies whether the new element is to be inserted as a child of the target element (the default), as the parent of the target element or as a sibling of the target element.

If inserting the copied element as a child, the 'offset' attribute specifies the number of nodes (not including comment nodes), from the top or bottom of the target element's list of children, in which to insert the new element. A negative value specifies up towards the first child. A positive value specifies down towards the last child. If there are fewer nodes than specified by the 'offset' attribute, the element will be inserted as either the first child or the last child. The 'from' attribute specifies whether the 'offset' is relative to the top (first child) or bottom (last child). The 'preserveTargetChildren', and 'preserveTargetEvents' and 'preserveTargetAttributes' attributes are ignored.

If inserting the copied element as the parent, the 'offset', 'from', 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' attributes are ignored.

If inserting the copied element as a sibling, the 'from', 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' attributes are ignored. The 'offset' specifies the number of nodes (not including comment nodes) before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert the new element. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child of the parent.

Other examples of behavior elements will be described below.

The following is an example of SVG markup used in association with an embodiment of the DOM manipulation system 20.

```
<?xml version="1.0"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
"http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-st
ylable.dtd">
<svg xmlns:dsvg="http://dsvg.corel.com/behaviors"
onload="dsvgInit(evt)" width="256" height="256">
  <desc>Example copyElement01 - copies the circle when the circle
is clicked.</desc>

  <script type="text/ecmascript" xlink:href="dsvg/dSVG.js"/>
  <script type="text/ecmascript"
xlink:href="dsvg/copyElement.js"/>
  <script type="text/ecmascript"
xlink:href="dsvg/setAttribute.js"/>

      <circle id="myCircle" cx="128" cy="34" r="16" fill="black">
           <dsvg:copyElement event="onclick"
           newElementID="myCircle2" sourceElementID="myCircle"
           targetElement="myCircle" insertAs="sibling"
           offset="1"/>
```

```
            <dsvg:setAttribute event="onclick"
            elementID="myCircle2" name="cy" value="192"/>
        </circle>
    </svg>
```

Figure 3 shows a representation of a browser 11 output display of the above SVG file. Figure 3 shows a circle 31 in the top half of the browser window 30. Figure 4 shows a node representation of a DOM 40 of the circle on the top half of a web page as shown in Figure 3. The DOM contains a node for the root 'svg' element and a node for the circle 43 element. The DOM representation also contains script nodes 41 and the <dsvg:copyElement> and <dsvg:setAttribute> behavior elements 42 added as children of the <circle> node 43. The <dsvg:copyElement> behavior element 42 contains the prefix "dsvg:" in the name 23 and contains an 'event' attribute 24 equal to "onclick". If a user clicks on the circle 31 in the top half of the browser window 30, a second circle 52 is created below the first circle 31 as shown in Figure 5.

The structure of behavior elements 22 allows them to be inserted in an XML file, such as an SVG file. Behavior elements 22 may be executed at load time (i.e., when the viewer 13 receives the file and creates the DOM), in response to an event, or grouped together as children of a <dsvg:action> container element, which is associated to an element via a <dsvg:listener> element.

In order for behavior elements 22 to be executed at load time, the behavior elements have their 'event' attribute set to "onload", which is the event created by the viewer 13 after the DOM has been built and all of the script is loaded into memory. All of these behavior elements are executed by the viewer's 13 script interpreter at load time, sequentially in the order in which they appear in the DOM.

Figure 6 shows an example of a method of manipulating a DOM 15 at load time (60) in accordance with the DOM manipulation system 20. At load time, after the viewer has finished building the DOM and loading the script, the method (60) begins with the initialization function being run by the viewer's script interpreter, which determines if the first DOM element is a designated element (61). If a designated element is found (62), then the name of the function associated with the designated element is automatically generated (63) (in accordance with a predetermined function naming convention) and called (64). Preferably, the predetermined function naming convention is similar to the predetermined element naming convention. If a designated element is not found (62), or after a generated function is called (64), the method determines if there are more elements in the DOM to search (65). If there are more elements in the DOM (65), the method determines if the next sibling element is a designated element (66). The process is repeated until all elements in the DOM are searched. Once there are no more elements in the DOM to search (65), then the method is done (66).

The method described in Figure 6 will be described using the above example of the SVG file and Figures 3, 4, and 5. The viewer's 13 script interpreter will execute the dsvgInit() function at load time, which will traverse each node in the DOM 40, searching for elements whose names 23 begin with the "dsvg:" prefix. The <dsvg:copyElement> node is found. Since it is a child of a 'circle' element that does not begin with the "dsvg:" prefix, the value of the node's 'event' attribute is retrieved (via the DOM's getAttribute() method) and found to be equal to "onclick". An event listener is placed on the parent 'circle' element, so that if the 'circle' element generates an "onlick" event, that event will be dispatched to the processActions(evt) handler function. The initialization function then looks fore more elements that begin with the "dsvg:" prefix and finds the <dsvg:setAttribute> node. Its 'event' attribute is determined to be "onclick", and so the initialization function attempts to place an "onclick" event listener on the 'circle' element. But it discovers that the same event listener is already on the 'circle' element, so it does not add another. The initialization function does not find any more elements beginning with the "dsvg:" namespace, and so its job is finished.

When the user clicks on the circle, the viewer creates an "onclick" event, which the event listener hears and dispatches to the processActions(evt) handler function. This function determines, from the event object passed in via the "evt" parameter, that the "onclick" event occurred on the 'circle' element. It then searches all the children of the 'circle' element, looking for any elements that begin with the "dsvg:" prefix. It finds the <dsvg:copyElement> first, and dynamically generates the string,

```
dsvgCopyElement(element, evt)
```

which gets converted to a real function call, whose parameters include the <dsvg:copyElement> object itself ('element') as well as the event ('evt') object. That newly generated dsvgCopyElement() function is then called, expecting it to have been either included in the DOM manipulation system 20 as a script 26, or referenced from the original document. This dsvgCopyElement() function contains script 26 which first retrieves the values of the attributes of the object passed in via the 'element' parameter, using the DOM's getAttribute() method. Since the value of the 'event' attribute is "onclick", which is the same as the event that triggered this function to be called, the function does not abort, but proceeds to use all of the information from the other attributes to copy the element via its script 26.

Once the dsvgCopyElement() function is done, the processActions() handler function searches for more elements that begin with the "dsvg:" prefix and finds the <dsvg:setAttribute> element. As before, the string "dsvgSetAttribute(element, evt)" is automatically generated and converted to a real function call, which gets called, whose script 26 gets run, resulting in the newly created 'circle' element's 'cy' attribute being set, causing the new circle 52 to be displayed in the bottom half of the document.

In this example, only the 'cy' attribute of the copy 52 of the circle was changed. However, many other modifications may occur to elements in a DOM.

In the example described above, the function was dynamically generated, i.e., a string was created, having the same prefix as the designated element (without the colon) and the same name as the designated element (except with the first letter capitalized) and with the designated element's object and the trigger event object passed in as two parameters. The script 26 or set of instructions for the operations of the generated function is stored in a predetermined format either in the document text file or in a separate text file on a file system or webserver, and is loaded into memory by the viewer at load time. Alternatively, the initialization function may search for elements that begin with the "dsvg:" prefix and, using an 'if' or 'switch' statement, determine the appropriate predetermined function to call, which again are expected to have been already loaded in memory by the viewer.

It is advantageous, though, for the function names to be generated dynamically, so that the main script file containing the initialization function 21 does not need to be updated whenever a new type of behavior element 22 has been created and is available for use.

As well, while the functions 26 that handle each type of behavior element 22 could be stored all in one file, it is advantageous to store them in separate files and reference them in the document only if their corresponding behavior element 22 is being used, so that only the code that is required is actually transmitted.

One way for a behavior element 22 to be executed in response to an event on a particular element is for the behavior element 22 to be inserted as a child of that particular element. The parent element can then be classified as the "observer element", since it has an event listener attached to it. When an event that is being listened for occurs on the observer element, the child behavior elements are then executed sequentially for each behavior element 22 that has an 'event' attribute value that matches the event that just occurred on the observer element.

Figure 7 shows an example of a method of manipulating a DOM 15 in response to an event (70) in accordance with the DOM manipulation system 20. The DOM manipulation system 20 is built on top of an event-driven architecture, such as SVG, and XML. Once an event occurs on an SVG element (i.e., the observer element), the method (70) begins with passing the event object to a handler function (71). The handler function determines if the first child element of the SVG element associated with the object is a designated element (72). If a designated element is found (73), then the handler function determines if the event attribute 24 of the designated element is equal to the event that has occurred (74). If the event attribute 24 of the designated element is equal to the event which triggered this method (70), then the name of the function associated with the designated element is automatically generated (75) (in accordance with a predetermined function naming convention) and called (76). Preferably, the predetermined function naming convention is similar to the predetermined element naming convention. If a designated element is not found (73), or if the event attribute 24 of the designated element does not match the trigger event (74), or after a generated function is called (76), the event handler determines if there are more child elements of the observer element to search (77). If there are more child elements of the observer element (77), the event handler determines if the next child is a designated element (78). Steps (73) to (78) are repeated until all child elements of the observer element are searched. Once there are no more child elements to search (77), then the handler function is done (79).

In another example shown below, clicking on the red circle will cause a new blue square to be created beside it:

```
<circle cx="10" cy="10" r="5" fill="red">
      <dsvg:createElement event="onclick"
newElementID="myRectangle" elementName="rect"/>
      <dsvg:setAttribute elementID="myRectangle"
name="x" value="20"/>
      <dsvg:setAttribute elementID="myRectangle"
name="y" value="5"/>
      <dsvg:setAttribute elementID="myRectangle"
name="width" value="10"/>
      <dsvg:setAttribute elementID="myRectangle"
name="height" value="10"/>
      <dsvg:setAttribute elementID="myRectangle"
name="fill" value="blue"/>
</circle>
```

Many manipulations may be performed by adding a plurality of behavior elements as children to an observer element in a DOM 15. Alternatively, the behavior elements 22 may be grouped as children of a <dsvg:action> element. The following is the syntax for the <dsvg:action> element:

```
<dsvg:action
      id="string"
      event="string"
/>
```

The <dsvg:acton> element is a container for behavior elements. The <dsvg:action> element gets associated to an observer element (e.g., a circle or button that gets clicked on) via the <dsvg:listener> element. This is useful because the

<dsvg:action> element and its children are not tied directly to the observer element, thus allowing them to be reused.

The <dsvg:action> element contains two attributes. The 'id' attribute allows the <dsvg:action> element to be referenced by a <dsvg:listener> element. The 'event' attribute specifies the event for which the observer element listens.

The <dsvg:action> element may then be associated with an observer element using a <dsvg:listener> element. The following is the syntax for the <dsvg:listener> element:

```
<dsvg:listener
      id="string"
      event="string"
      {observer="xpath" | observerFrameID="string"
      observerObjectID="string" observerDocID="string"
      observerElementID="string"}
      {handler="xpath" | handlerFrameID="string"
      handlerObjectID="string" handlerDocID="string"
      handlerElementID="string" |
      handlerFunction="string"}
/>
```

The <dsvg:listener> element listens for the specified event on the specified observer element and, if found, passes control to a handler element which will respond to the event. The handler element may be a behavior element 22 or a handler function. This is useful because the handler element (which may be an <action> container for many behavior elements 22 to be executed sequentially) is not tied directly to the observer element, thus allowing it to be reused.

The <dsvg:listener> element contains many attributes. The 'id' attribute allows this behavior element to be referenced later. The 'event' attribute specifies the event on the observer element to listen for. The 'observer' attribute specifies the xpath to the observer element, e.g., the element that gets clicked on. If the 'observer' attribute is specified, then the observerFrameID, observerObjectID, observerDocID and observerElementID attributes are ignored. The 'observerFrameID' attribute specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the observer element. If the 'observerFrameID' attribute is not provided, the current frame is assumed. The 'observerObjectID' attribute specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the observer element. If the 'observerObjectID' attribute is not provided, the current object is assumed. The 'observerDocID' attribute specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the observer element. If the 'observerDocID' attribute is not provided, the current document is assumed. The 'observerElementID' attribute specifies the 'id' attribute of the observer element.

The 'handler' attribute specifies the XPath to the handler element, e.g., the element that gets executed. If the 'handler' attribute is specified, then the handlerFrameID, handlerObjectID, handlerDocID and handlerElementID attributes are ignored. The 'handlerFrameID' attribute specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the handler element. If the 'handlerFrameID' attribute is not provided, the current frame is assumed. The 'handlerObjectID' attribute specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the handler element. If the 'handlerObjectID' attribute is not provided, the current object is assumed. The 'handlerDocID' attribute specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the handler element. If the 'handlerDocID' attribute is not provided, the current document is assumed. The 'handlerElementID' attribute specifies the 'id' attribute of the handler element. The 'handlerFunction' specifies the name of the script function (with any required variables) to be executed.

The following is the alternative syntax for the previous example of creating a blue square next to a red circle, whereby the behavior elements 22 are grouped as children of an <action> element:

```
<circle id="myCircle" cx="10" cy="10" r="5"
fill="red"/>

<dsvg:action id="createRectangle">
```

```
        <dsvg:createElement event="onclick"
newElementID="myRectangle" elementName="rect"/>
        <dsvg:setAttribute elementID="myRectangle"
name="x" value="20"/>
        <dsvg:setAttribute elementID="myRectangle"
name="y" value="5"/>
        <dsvg:setAttribute elementID="myRectangle"
name="width" value="10"/>
        <dsvg:setAttribute elementID="myRectangle"
name="height" value="10"/>
        <dsvg:setAttribute elementID="myRectangle"
name="fill" value="blue"/>
</dsvg:action>

<dsvg:listener event="onclick"
observerElementID="myCircle"
handlerID="createRectangle"/>
```

The DOM manipulation system 20 also allows for behavior attributes to be added to existing regular SVG or XML elements. Preferably, the behavior attributes have a name which follows a pre-determined naming convention. A general behavior attribute 'dsvg:behaviorAttribute' can be added to the element <existingElement>:

```
<existingElement
        attribute1="value"
        attribute2="value"
        dsvg:behaviorAttribute="value"
/>
```

Accordingly, a collection (or set) of behavior attributes may be added to the DOM manipulation system 20. Similarly, a corresponding collection (or set) of scripts may be added to the DOM manipulation system 20. The corresponding scripts comprise functions or instructions which are performed in association with the regular XML or SVG element.

The initialization file 21 may also search for attributes in elements that are not behavior elements 22. Scripts 26 may be created and associated with the 'dsvg' attribute in the same manner as with behavior elements. Script functions 26 for 'dsvg' attributes only operate on the object associated with the existing element to which a 'dsvg' attribute is added. A designer may add the 'dsvg' attribute in an SVG file, or any other XML file to be parsed by the viewer 13.

Figure 8 shows another example of an method of manipulating a DOM (80) in accordance with the DOM manipulation system 20. After a user (or designer) marks up an SVG file using the markup syntax of the DOM manipulation system and the SVG file is loaded into a viewer 13, the viewer 13 creates an "onload" event which is received by an <svg> element. The method (80) begins with the initialization function 21. A dsvgInit() initialization function 21 is called (81) by the viewer's script interpreter, which traverses the nodes of the DOM. The initialization function 21 determines if the first DOM element is a designated element (82). If a designated element is found (83) and the 'event' attribute of the designated element is set to "onload" (84), then the name of the function associated with the designated element is automatically generated (85) (in accordance with a predetermined function naming convention) and called (86). Preferably, the predetermined function naming convention is similar to the predetermined element naming convention. If a designated element is not found (83), the initialization function 21 determines if the regular SVG element contains any designated attributes (87). If any designated attributes are found (87), then the names of the functions associated with the designated attributes are automatically generated (88) (again, in accordance with a predetermined function naming convention) and called (89).

If a designated attribute is not found (87), then the initialization file 21 determines if the regular SVG element has any child elements (90). If the regular SVG element has a child element (90) and the child element is a designated element 22 (91), then the initialization file 21 determines the value of the designated element's 'event' attribute (i.e., the event that will trigger the execution of the designated element's associated function) and adds that event listener to the parent SVG element (92) (via the addEventListener() DOM API). If the child element is not a designated element 22 (91), then the initialization file 21 determines if there are any other children of the regular SVG element (93). If there are more children (93), then the initialization file searches the next child of the regular SVG element (94). Steps (91) to (94) repeat until there are no more children of the regular SVG element.

If there are no more children of the regular SVG element (93), or after a generated function is called (86, 89), or if the event attribute of a designated element is not equal to "onload" (84), or there are no more child elements in a regular SVG element to search (90), the initialization file 21 determines if there are more elements in the DOM to search (95). If there are more elements in the DOM (95), the initialization file determines if the next sibling element is a designated element (96). Steps (83) to (96) are repeated until all elements in the DOM are searched. Once there are no more elements in the DOM to search (95), then the initialization function 21 is done and the viewer 13 waits for an event to occur (97).

Once an event occurs on an SVG element (i.e., the observer element), that event object is passed to any handler function with which it has been associated (98). The handler function determines if the first child of the observer element is a designated element (99). If a designated element is found (100), then the handler function determines if the event attribute 24 of the designated element is equal to the event that has occurred (101). If the event attribute 24 of the designated element is equal to the event (101), then the name of the function associated with the designated element is automatically generated (102) (in accordance with a predetermined function naming convention) and called (103). Preferably, the predetermined function naming convention is similar to the predetermined element naming convention. If a designated element is not found (100), or if the event attribute 24 of the designated element does not match the trigger event (101), or after a generated function is called (103), the event handler determines if there are more child elements of the observer element to search (104). If there are more child elements of the observer element (104), the method determines if the next child is a designated element (105). Steps (100) to (105) are repeated until all child elements of the observer element are searched. Once there are no more child elements to search (104), then the event handler function is done and the viewer waits for another event to occur (97).

Referencing Attributes

To create an application, a designer often desires to reference the current value of another element's attributes. The system 20 allows for the following syntax to perform this reference:

%frameID.objectID.docID.elementID.childElementID@attributeName%

If not specified, the 'frameID', 'objectID' and 'docID' are assumed to be the current frame, object and document. In the following example, hovering the mouse over circle1 causes circle3 to turn blue and hovering over circle2 causes circle3 to turn red.

```
<circle id="myCircle1" x="10" y="10" r="5" fill="red">
     <dsvg:setAttribute event="onmouseover"
elementID="myCircle3" value="%myCircle2@fill%">
</circle>

<circle id="myCircle2" x="20" y="10" r="5"
fill="blue">
     <dsvg:setAttribute event="onmouseover"
elementID="myCircle3" value="%myCircle1@fill%">
</circle>

<circle id="myCircle2" x="15" y="20" r="5"
fill="green">
```

The system 20 also allows for parenthesis and mathematical operators within the % % expression. For example, in the markup:

```
<dsvg:button id="button" x="50" y="50" label="foo"
toggle="true" group="pickPanGroup"
xlink:href="skinButton_Windows.svg#skinButton"
selected="true" />
```

```
<dsvg:button id="button_2" x="200" y="200" label="foo"
toggle="true" group="pickPanGroup"
xlink:href="skinButton_Windows.svg#skinButton"
selected="true" />
```

an attribute "foo %button_(button@x - 48).x + 14 * button@y% bar%button@x%" is parsed as follows:

```
'foo %button_(button@x - 48)@x + 14 * button@y% bar%button@x/2%'
-> 'foo %button_(50 - 48).x + 14 * button@y% bar%button@x/2%'
-> 'foo %button_(2).x + 14 * button@y% bar%button@x/2%'
-> 'foo %button_2.x + 14 * button@y% bar%button@x/2%'
-> 'foo %200 + 14 * 50% bar%button@x/2%'
-> 'foo %200 + 700% bar%button@x/2%'
-> 'foo 900 bar%button@x/2%'
-> 'foo 900 bar25'
```

Figure 9 shows an example of a method of creating an element for manipulating a DOM (200) in accordance with the DOM manipulation system 20. The method (200) begins with categorizing low level actions into behavior groupings (201). Next, common attributes of a behavior grouping are determined (202). Next, a behavior element 22 having the common attributes of the behavior grouping is created (203). If there are more behavior groupings categorized in step (201), then steps (202) to (204) are repeated. If there are no more behavior grouping, then the method is done (205). Other steps may be added to this method (200). The method (200) may be used to create a plurality of behavior elements 22.

Figure 10 shows another example of a method of creating an element for manipulating a DOM (210) in accordance with the DOM manipulation system 20. The method (210) begins with organizing low level actions into groups of similar actions (211). Next behavior names are designated to the groupings (212). Next variations of a grouping are analyzed to determine common attributes of the grouping (213). A list of attributes needed to perform the variations is compiled (214). A behavior element 22 is then created. A name is assigned to the behavior element 22 pursuant to a naming convention (215). Next, the common attributes 25 of the behavior grouping are assigned to the behavior element 22 (216). Finally, a set of instructions or one or more functions (script 26) are created to be used by the behavior element 22 (217). The behavior element may be stored in an independent file (218). Once all behavior elements 22 have been created (219), the method is done (220). Default settings may be initiated for the behavior elements if desired.

There are many advantages to the DOM manipulation system 20. The system 20 enables web designers with little or no programming skills to create dynamic, interactive web applications. It also benefits experienced programmers, allowing them to write stable, robust applications extremely quickly (RAD–Rapid Application Developement)–much more quickly than via script. Because the DOM manipulation system 20 uses an XML markup language (as opposed to script libraries), the attributes and data and even the elements themselves can be made to be data-driven at run-time, using (at design-time) existing or new software that allows one to visually map input XML markup to output XML markup, resulting in an XSLT code (or any other language useful for XML transformations) which will actually modify the DOM Manipulation markup based on the input XML data/markup.

The DOM manipulation system 20 can also be natively-implemented, accessing the exposed DOM API's in the same manner as the script implementation. A native implementation could be much faster because unlike script, which gets interpreted at run-time, native code (e.g. C++ or C) gets interpreted at compile time and gets optimized by the compiler. The natively-implemented DOM manipulation system 20 could also access any unexposed, lower-level object model API's directly, rather than the exposed higher-level DOM API's, which could further improve performance. If natively implemented, the amount of data needed to be transferred is greatly reduced, since there is no script that needs to be transmitted, which is especially beneficial for wireless devices with low bandwidth and small memory. Using a markup language for the DOM Manipulation behavior elements is also beneficial because it allows for the possibility of further reduced the file size by creating a binary version of the markup language that employs opcodes–predetermined arrangements of bits (1's and 0's) that correspond to particular element names and attributes. Unlike textual markup, which must be parsed (compared to predetermine strings/text to establish the meaning of the text) in order to create the DOM, binary opcodes can be compared to identical binary opcodes, which is much faster than string comparisons, in order to build the DOM much faster.

The DOM manipulation system 20 abstracts the DOM API's to create a more direct linkage between the syntax and the intent of the author, thus making it easier to create SVG based applications. The behavior elements <setStyle>, <setClass> and <setTransform> make modifying the 'style', 'class' and 'transform' attributes much easier for the designer. These attributes are typically difficult to modify with script because they do not map directly to one value, but instead are composed of a string of separate properties or property-value pairs themselves. For instance, if a designer wishes to set an element's stroke-width to 1, the designer cannot simply set a "stroke-width" attribute because it does not exist. The designer would have to set the 'style' attribute to "stroke:1". However the 'style' attribute may already have had more style properties defined, such as style="stroke-width:2;fill:red;opacity:0.5". Therefore, if the designer simply sets the 'style' attribute to "stroke:1", then the designer would accidently remove all the other style properties already defined. Therefore, a designer needs to first get the value of the 'style' attribute, parse it, determine if it already has the property the designer wants to set, set it or replace it, and write the new delimited string. These steps are all performed with the <setStyle> behavior element.

Modifying a 'transform' attribute has similar problems and is more difficult with its syntax of transform="matrix(a b c d e f) translate(x [y]) scale(sx [sy] rotate(angle [cx cy]) skewX(angle) skewY(angle))". Further complicating matters, the final transformation depends on the order of these individual transformations. Also, applying a scale factor to an element has the effect of scaling that element's coordinates, thus causing the element's location on the screen to change. Therefore, mathematical calculations are required to determine the transformation needed to preserve the element's centre or edge coordinates. These requirements are handled by the <setTransform> behavior element.

Thus, the <setStyle>, <setClass> and <setTransform> behavior elements, as well as the <setLink> and <setStyleSheet> behavior elements, effectively abstract the designer from having to understand the details of the syntax of SVG. The % % syntax described above enables any element to reference any attribute of any element, thus enabling the creation of interactive, dynamic, client-side data driven web applications.

The DOM manipulation system 20 according to the present invention may be implemented by any hardware, software or a combination of hardware and software having the above described functions. The software code, either in its entirety or a part thereof, may be stored in a computer readable memory. Further, a computer data signal representing the software code which may be embedded in a carrier wave may be transmitted via a communication network. Such a computer readable memory and a computer data signal are also within the scope of the present invention, as well as the hardware, software and the combination thereof.

While particular embodiments of the present invention have been shown and described, changes and modifications may be made to such embodiments without departing from the true scope of the invention.

Listing of Behavior Elements 22

Some examples of behavior elements 22, in accordance with the DOM manipulation system 20, are provided below. The examples provide a syntax, a description and attributes of the behavior elements. Other behavior element 22 may be created. The provided behavior elements 22 are examples of one implementation of a DOM manipulation markup language.

Generic XML DOM:

<action>

Syntax:

<action

id="*string*"

event="*string*"

/>

Description:

A container for behavior elements 22. It can either be used as a container for behavior elements having the same event attribute value, in which case the 'event' attribute must be defined for the 'action' element but not for the child behavior elements, or it can be associated to an observer element (e.g., a circle or button that gets clicked on) via the <listener> element, in which case its 'event' attribute is not required. The latter is useful because the <action> element and its children are not tied directly to the observer element, thus allowing them to be reused.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.

<copyElement>

Syntax:

```
<copyElement
        id="string"
        event="string"
        newElementID="string"
        {source="xpath" | sourceFrameID="string" sourceObjectID="string"
sourceDocID="string" sourceElementID="string"}
        {target="xpath" | targetFrameID="string" targetObjectID="string"
targetDocID="string" targetElementID="string"}
        insertAs="{child | parent | sibling | replacement}"
        offset="signed integer"
        from="{top | bottom}"
        copyChildren="{true | false}"
        copyEvents="{true | false}"
        copyAttributes="{all | none | attr1;attr2;...attrN}"
        preserveTargetChildren="{true | false}"
        preserveTargetEvents="{true | false}"
        preserveTargetAttributes="{all | none | attr1;attr2;...attrN}"
/>
```

Description:

Creates a copy of an existing element and inserts it into the DOM at the desired location.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'newElementID' specifies the value of the newly created element's 'id' attribute.
- 'source' is the xpath pointing to the element to be copied. If provided, 'sourceFrameID', 'sourceObjectID', 'sourceDocID' and 'sourceElementID' are ignored.
- 'sourceFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element to be copied. If not provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element to be copied. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element to be copied. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceElementID' specifies the 'id' attribute of the element to be copied. If 'source' is provided, this attribute is ignored.
- 'target' is the xpath pointing to the element at which to insert the new element. If provided, 'targetFrameID', 'targetObjectID', 'targetDocID' and 'targetElementID' are ignored.
- 'targetFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to place the new element. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'targetObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the new element. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.

- 'targetDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to place the new element. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'targetElementID' specifies the 'id' attribute of the element at which to insert the new element. If 'target' is provided, this attribute is ignored.
- 'insertAs' specifies whether the new element is to be inserted as a child of the target element (the default), as the parent of the target element or as a sibling of the target element.
- If inserting as a child:
    - 'offset' specifies the number of nodes (not including comment nodes) from the top or bottom at which to insert the new element. A negative value specifies up towards the first child. A positive value specifies down towards the last child. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child.
    - 'from' specifies whether 'offset' is relative to the top (first child) or bottom (last child).
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as the parent:
    - 'offset' is ignored.
    - 'from' is ignored.
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a sibling:
    - 'offset' specifies the number of nodes (not including comment nodes) before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert the new element. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child of the parent.
    - 'from' is ignored.

    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a replacement (in which the target element is removed and replaced):
    - 'preserveTargetChildren' specifies whether to copy the target element's children (true) or not (false).
    - 'preserveTargetEvents' specifies whether to copy the target element's events (e.g. onmouseover) (true) or not (false).
    - 'preserveTargetAttributes' specifies that all of the target element's attributes, none of them, or a list of specific attributes should be copied.
- 'copyChildren' specifies whether to copy the source element's children (true) or not (false).
- 'copyEvents' specifies whether to copy the source element's events (e.g., onmouseover) (true) or not (false).
- 'copyAttributes' specifies that all of the source element's attributes, none of them, or a list of specific attributes should be copied.

<createCDATASection>

Syntax:

<createCDATASection
        id="*string*"
        event="*string*"
        {target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*"
elementID="*string*"}
        data="*string*"
/>

Description:

Creates a CDATA section, with data, as a child of the specified element.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.

- 'target' is the xpath pointing to the element that is to be the parent of the CDATA block. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g. an HTML <frame> element) in which to find the element that is to be the parent of the CDATA block. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element that is to be the parent of the CDATA block. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element that is to be the parent of the CDATA block. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element that is to be the parent of the CDATA block. If 'target' is provided, this attribute is ignored.
- 'data' specifies the text within the CDATA section.

<createDocument>

Syntax:

<createDocument
    id="*string*"
    event="*string*"
    {target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*"}
    namespaceURI="*string*"
    qualifiedName="*string*"
    fragment="{true | <u>false</u>}
/>

Description:

Creates a new XML document or documentFragment.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location at which the new document is to be inserted. If provided, 'frameID', 'objectID' and 'docID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to place the new document. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the new document. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the ID that the newly created document should have. If not provided (and 'target' is also not provided), no document will be created.. If 'target' is provided, this attribute is ignored.
- 'namespaceUR'I specifies the URI of the namespace, e.g., http://www.w3.org/TR/2000/03/WD-SVG-20000303/DTD/svg-20000303-stylable.dtd.
- 'qualifiedName' specifies the local name of the root element, e.g., "svg".
- 'fragment' specifies whether to create a document (false) or a documentFragment (true). A documentFragment is a lightweight document used for such purposes as constructing or rearranging elements to be inserted back into the real document, or storing XML data of a non-native format.

<createElement>

Syntax:

<createElement
    id="*string*"
    event="*string*"
    newElementID="*string*"
    elementName="*string*"
    attributes="*string*"
    ns="*string*"

{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}
insertAs="{<u>child</u> | parent | sibling | replacement}"
offset="*signed integer*"
from="{<u>top</u> | bottom}"
preserveTargetChildren="{<u>true</u> | false}"
preserveTargetEvents="{<u>true</u> | false}"
preserveTargetAttributes="{<u>all</u> | none | *attr1;attr2;...attrN}*"
/>

Description:
Creates a new element and inserts it in the desired location in the DOM.
Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'newElementID' specifies the value of the newly created element's 'id' attribute.
- 'elementName' specifies the name of the element to be created (e.g., rect (or rectangle), ellipse, etc.).
- 'attributes' is a space-separated string containing all of the attributes for the element (e.g., attributes="x='10' y='10' width='50' height='50' fill='red'"). This attribute is optional–the attributes could instead be set afterwards using <setAttribute>. Note that the apostrophes must be used instead of quotes within the string, since a quote would be interpreted as the closing-quote of the 'attributes' value, which would cause a parsing error.
- 'ns' specifies the namespace of the element. The namespace is prefixed to the element name, separated by a colon. If the namespace is not defined in the document's root element, it will be defined in the new element.
- 'target' is the xpath pointing to the location at which the new element is to be inserted. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to place the new element. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.

- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the new element. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to place the new element. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element at which to insert the new element. If 'target' is provided, this attribute is ignored.
- 'insertAs' specifies whether the new element is to be inserted as a child of the target element (the default), as the parent of the target element or as a sibling of the target element.
- If inserting as a child:
    - 'offset' specifies the number of nodes (not including comment nodes) from the top or bottom at which to insert the new element. A negative value specifies up towards the first child. A positive value specifies down towards the last child. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child.
    - 'from' specifies whether 'offset' is relative to the top (first child) or bottom (last child).
    - 'preserveTargetChildren', 'preserveEvents' and 'preserveAttributes' are ignored.
- If inserting as the parent:
    - 'offset' is ignored.
    - 'from' is ignored.
    - 'preserveTargetChildren', 'preserveEvents' and 'preserveAttributes' are ignored.
- If inserting as a sibling:
    - 'offset' specifies the number of nodes (not including comment nodes) before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert the new element. If there are fewer nodes than

  - specified by 'offset', the element will be inserted as either the first child or the last child of the parent.
  - 'from' is ignored.
  - 'preserveTargetChildren', 'preserveEvents' and 'preserveAttributes' are ignored.
- If inserting as a replacement (in which the target element is removed and replaced):
  - 'preserveTargetChildren' specifies whether to copy the target element's children (true) or not (false).
  - 'preserveTargetEvents' specifies whether to copy the target element's events (e.g., onmouseover) (true) or not (false).
  - 'preserveTargetAttributes' specifies that all of the target element's attributes, none of them, or a list of specific attributes should be copied.

<createEvent>

Syntax:

<createEvent

id="*string*"

event="*string*"

eventName="*string*"

{source="*xpath*" | sourceFrameID="*string*" sourceObjectID="*string*" sourceDocID="*string*" sourceElementID="*string*"}

{target="*xpath*" | targetFrameID="*string*" targetObjectID="*string*" targetDocID="*string*" targetElementID="*string*"}

eventPhase="*unsigned integer*"

bubbles="{<u>true</u> | false}"

cancelable="{true | false}"

timeStamp="*string*"

stopPropagation="{true | <u>false</u>}"

preventDefault="{true | <u>false</u>}"

/>

Description:

Creates an event and dispatches (sends) it to the desired target.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'eventName' specifies the type of the event, e.g., mouseover. Note that only events that are supported by the software (e.g., the SVG Viewer) can be created.
- 'source' specifies the xpath to the element that the target will "think" created the event. If not specified, and sourceElementID is also not specified, the source will be the <createElement> itself. If specified, then sourceFrameID, sourceObjectID, sourceDocID and sourceElementID are ignored.
- 'sourceFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element that the target will "think" created the event. If not provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element that the target will "think" created the event. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element that the target will "think" created the event. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceElementID' specifies the 'id' attribute of the element that the target will "think" created the event. If 'source' is provided, this attribute is ignored.
- 'target' specifies the xpath to the element that the event is being dispatched to. If specified, then targetFrameID, targetObjectID, targetDocID and targetElementID are ignored.
- 'targetFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element that the event is being dispatched to. If not

provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.

- 'targetObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element that the event is being dispatched to. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.
- 'targetDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element that the event is being dispatched to. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.
- 'targetElementID' specifies the 'id' attribute of the element that the event is being dispatched to. If 'source' is provided, this attribute is ignored.
- 'eventPhase' specifies which phase of event flow is currently being evaluated.
- 'bubbles' specifies whether the event can bubble (true) or not (false).
- 'cancelable' specifies whether the event can have its default actions prevented (true) or not (false).
- 'timeStamp' specifies the time (in milliseconds since the epoch) at which the event was created. If not supplied, the current system time is used. If not available, zero is used. Examples of epoch time are the time of the system start or 0:0:0 UTC 1st January 1970.
- 'stopPropagation' prevents further propagation of an event during event flow. If true, the event will complete its dispatch to all listeners and then cease propagating through the tree.
- 'preventDefault' specifies (if true) that the event is to be cancelled, so that any default action normally taken by the implementation as a result of the event will not occur. This has no effect, however, on non-cancelable events, i.e., it has no effect if 'cancelable' is false.

<createProcessingInstruction>

Syntax:

<createProcessingInstruction

id="*string*"

event="*string*"

data="*string*"

/>

Description:

Creates a processingInstruction for the document, e.g., <? xml version="1.0"?>

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'data' specifies the processingInstruction, e.g., xml version=" 1.0".

<dispatchEvent>

<dispatchEvent

id="*string*"

event="*string*"

{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}

/>

Description:

Dispatches whatever event triggered this action-element to the desired target.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' specifies the xpath to the element that the event is being dispatched to. If specified, then targetFrameID, targetObjectID, targetDocID and targetElementID are ignored.
- 'targetFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element that the event is being dispatched to. If not provided, the current frame is assumed.

- 'targetObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element that the event is being dispatched to. If not provided, the current object is assumed.
- 'targetDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element that the event is being dispatched to. If not provided, the current document is assumed.
- 'targetElementID' specifies the 'id' attribute of the element that the event is being dispatched to.

<listener>

Syntax:

<listener
    id="*string*"
    event="*string*"
    {observer="*xpath*" | observerFrameID="*string*" observerObjectID="*string*" observerDocID="*string*" observerElementID="*string*"}
    {handler="*xpath*" | handlerFrameID="*string*" handlerObjectID="*string*" handlerDocID="*string*" handlerElementID="*string*" | handlerFunction="*string*"}
/>

Description:

Listens for the specified event on the specified observer element and, if found, passes control to the handler element (such as a behavior element 22) or handler function for processing. This is useful because the handler element (which may be an <action> container for many behavior elements 22 to be executed sequentially) is not tied directly to the observer element, thus allowing it to be reused.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event on the observer element to listen for.
- 'observer' specifies the xpath to the observer element, e.g. the element that gets clicked on. If specified, then observerFrameID, observerObjectID, observerDocID and observerElementID are ignored.

- 'observerFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the observer element. If not provided, the current frame is assumed.
- 'observerObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the observer element. If not provided, the current object is assumed.
- 'observerDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the observer element. If not provided, the current document is assumed.
- 'observerElementID' specifies the 'id' attribute of the observer element.
- 'handler' specifies the xpath to the handler element, e.g., the element that gets executed. If specified, then handlerFrameID, handlerObjectID, handlerDocID and handlerElementID are ignored.
- 'handlerFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the handler element. If not provided, the current frame is assumed.
- 'handlerObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the handler element. If not provided, the current object is assumed.
- 'handlerDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the handler element. If not provided, the current document is assumed.
- 'handlerElementID' specifies the 'id' attribute of the handler element.
- 'handlerFunction' specifies the name of the script function (with any required variables) to be executed.

<loadXML>

Syntax:

<loadXML

    id="*string*"

    event="*string*"

{source="*xpath*" | xlink:href="*URL*"}
{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}
insertAs="{<u>child</u> | parent | sibling | replacement}"
offset="*signed integer*"
from="{<u>top</u> | bottom}"
preserveTargetChildren="{<u>true</u> | false}"
preserveTargetEvents="{<u>true</u> | false}"
preserveTargetAttributes="{<u>all</u> | none | *attr1;attr2;...attrN}*"
/>

Description:

Loads a document or fragment (element, possibly with children) and either inserts it into the desired location of the DOM or into a new documentFragment (a lightweight document used for such purposes as constructing or rearranging elements to be inserted back into the real document, or storing XML data of a non-native format).

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'source' specifies the xpath to a document or element within a document in an external file. If provided, 'xlink:href' is ignored.
- 'xlink:href' specifies the URL to a document or element within a document in an external file. If 'source' is provided, this attribute is ignored.
- 'target' is the xpath pointing to wherever the document or fragment is to be placed. If provided, 'targetFrameID', 'targetObjectID', 'targetDocID' and 'targetElementID' are ignored.
- 'targetFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to place the new document or fragment. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'targetObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the new document or fragment. If not

provided, the current object is assumed. If 'target' is provided, this attribute is ignored.

- 'targetDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) or documentFragment in which to place the new document or fragment. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored. Note that if the XML data being loaded is of a different grammar than the current document, inserting it into the current document could cause problems, depending on what the XML Viewer allows. It is safer to provide a 'targetDocID', which will be the ID of the documentFragment used to store this XML data. The data can then be referenced by any other behavior element 22, such as <setAttribute>, as described above.
- 'targetElementID' specifies the 'id' attribute of the element at which to insert the new fragment. If 'target' is provided, this attribute is ignored. If an entire document is being loaded, this attribute will be ignored.
- 'insertAs' specifies whether the new fragment is to be inserted as a child of the target element (the default), as the parent of the target element or as a sibling of the target element. If an entire document is being loaded or if 'targetDocID' exists but 'targetElementID' does not, 'insertAs' is set to "replacement", which causes any document or documentFragment with an ID of 'targetDocID' to be replaced with the new document or, if no such document exists, causes a new documentFragment to be created with an ID of 'targetDocID'.
- If inserting as a child:
    - 'offset' specifies the number of nodes (not including comment nodes) from the top or bottom at which to insert the new fragment. A negative value specifies up towards the first child. A positive value specifies down towards the last child. If there are fewer nodes than specified by 'offset', the fragment will be inserted as either the first child or the last child. If an entire document is being loaded, or if 'targetDocID' exists but 'targetElementID' does not, this attribute is ignored.
    - 'from' specifies whether 'offset' is relative to the top (first child) or bottom (last child). If an entire document is being loaded, or if

    'targetDocID' exists but 'targetElementID' does not, this attribute is ignored.
  - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as the parent:
  - 'offset' is ignored.
  - 'from' is ignored.
  - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a sibling:
  - 'offset' specifies the number of nodes (not including comment nodes) before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert the new element. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child of the parent. If an entire document is being loaded, or if 'targetDocID' exists but 'targetElementID' does not, this attribute is ignored.
  - 'from' is ignored.
  - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a replacement (to an element, not to a document or documentFragment), in which the target element is removed and replaced:
  - 'preserveTargetChildren' specifies whether to copy the target element's children (true) or not (false).
  - 'preserveTargetEvents' specifies whether to copy the target element's events (e.g., onmouseover) (true) or not (false).
  - 'preserveTargetAttributes' specifies that all of the target element's attributes, none of them, or a list of specific attributes should be copied.

<modifyEvent>

Syntax:

```
<modifyEvent
        id="string"
        event="string"
        eventName="string"
        {source="xpath" | sourceFrameID="string" sourceObjectID="string"
sourceDocID="string" sourceElementID="string"}
        {target="xpath" | targetFrameID="string" targetObjectID="string"
targetDocID="string" targetElementID="string"}
        eventPhase="string"
        bubbles="{true | false}"
        cancelable="{true | false}"
        timeStamp="string"
        stopPropagation="{true | false}"
        preventDefault="{true | false}"
/>
```

Description:

Modifies whatever event triggered this action-element and dispatches (sends) it to the desired target.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'eventName' specifies the type of the event, e.g. mouseover. Note that only events that are supported by the software (e.g., the SVG Viewer) can be created.
- 'source' specifies the xpath to the element that the target will "think" created the event. If not specified, and sourceElementID is also not specified, the source will be the <createElement> itself. If specified, then sourceFrameID, sourceObjectID, sourceDocID and sourceElementID are ignored.
- 'sourceFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element that the target will "think" created the event. If not provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.

- 'sourceObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element that the target will "think" created the event. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element that the target will "think" created the event. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.
- 'sourceElementID' specifies the 'id' attribute of the element that the target will "think" created the event. If 'source' is provided, this attribute is ignored.
- 'target' specifies the xpath to the element that the event is being dispatched to. If specified, then targetFrameID, targetObjectID, targetDocID and targetElementID are ignored.
- 'targetFrameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element that the event is being dispatched to. If not provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.
- 'targetObjectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element that the event is being dispatched to. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.
- 'targetDocID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element that the event is being dispatched to. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.
- 'targetElementID' specifies the 'id' attribute of the element that the event is being dispatched to. If 'source' is provided, this attribute is ignored.
- 'eventPhase' specifies which phase of event flow is currently being evaluated.
- 'bubbles' specifies whether the event can bubble (true) or not (false).
- 'cancelable' specifies whether the event can have its default actions prevented (true) or not (false).

- 'timeStamp' specifies the time (in milliseconds since the epoch) at which the event was created. If not supplied, the current system time is used. If not available, zero is used. Examples of epoch time are the time of the system start or 0:0:0 UTC 1st January 1970.
- 'stopPropagation' prevents further propagation of an event during event flow. If true, the event will complete its dispatch to all listeners and then cease propagating through the tree.
- 'preventDefault' specifies (if true) that the event is to be cancelled, so that any default action normally taken by the implementation as a result of the event will not occur. This has no effect, however, on non-cancelable events, i.e., it has no effect if 'cancelable' is false.

<moveElement>

Syntax:

<moveElement

id="*string*"

event="*string*"

{source="*xpath*" | sourceFrameID="*string*" sourceObjectID="*string*" sourceDocID="*string*" sourceElementID="*string*"}

{target="*xpath*" | targetFrameID="*string*" targetObjectID="*string*" targetDocID="*string*" targetElementID="*string*"}

insertAs="{<u>child</u> | parent | sibling | replacement}"

offset="*signed integer*"

from="{<u>top</u> | bottom}"

preserveTargetChildren="{<u>true</u> | false}"

preserveTargetEvents="{<u>true</u> | false}"

preserveTargetAttributes="{<u>all</u> | none | *attr1;attr2;...attrN}*"

/>

Description:

Moves an existing element to a desired location in the DOM.

Attributes:

- ‘id’ allows this action element to be referenced later.
- ‘event’ specifies the event that triggers this action.
- ‘source’ is the xpath pointing to the element to be moved. If provided, ‘sourceFrameID’, ‘sourceObjectID’, ‘sourceDocID’ and ‘sourceElementID’ are ignored.
- ‘sourceFrameID’ specifies the ‘id’ attribute of the frame (e.g., an HTML <frame> element) in which to find the element to be moved. If not provided, the current frame is assumed. If ‘source’ is provided, this attribute is ignored.
- ‘sourceObjectID’ specifies the ‘id’ attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element to be moved. If not provided, the current object is assumed. If ‘source’ is provided, this attribute is ignored.
- ‘sourceDocID’ specifies the ‘id’ attribute of the document (e.g., an SVG or XHTML document) in which to find the element to be moved. If not provided, the current document is assumed. If ‘source’ is provided, this attribute is ignored.
- ‘sourceElementID’ specifies the ‘id’ attribute of the element to be moved. If ‘source’ is provided, this attribute is ignored.
- ‘target’ is the xpath pointing to the element at which to insert the element. If provided, ‘targetFrameID’, ‘targetObjectID’, ‘targetDocID’ and ‘targetElementID’ are ignored.
- ‘targetFrameID’ specifies the ‘id’ attribute of the frame (e.g., an HTML <frame> element) in which to place the element. If not provided, the current frame is assumed. If ‘target’ is provided, this attribute is ignored.
- ‘targetObjectID’ specifies the ‘id’ attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the element. If not provided, the current object is assumed. If ‘target’ is provided, this attribute is ignored.
- ‘targetDocID’ specifies the ‘id’ attribute of the document (e.g., an SVG or XHTML document) in which to place the element. If not provided, the current document is assumed. If ‘target’ is provided, this attribute is ignored.
- ‘targetElementID’ specifies the ‘id’ attribute of the element at which to insert the element. If ‘target’ is provided, this attribute is ignored.

- 'insertAs' specifies whether the element is to be inserted as a child of the target element (the default), as the parent of the target element or as a sibling of the target element.
- If inserting as a child:
    - 'offset' specifies the number of nodes (not including comment nodes) from the top or bottom at which to insert the element. A negative value specifies up towards the first child. A positive value specifies down towards the last child. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child.
    - 'from' specifies whether 'offset' is relative to the top (first child) or bottom (last child).
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as the parent:
    - 'offset' is ignored.
    - 'from' is ignored.
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a sibling:
    - 'offset' specifies the number of nodes (not including comment nodes) before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert the element. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child of the parent.
    - 'from' is ignored.
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a replacement (in which the target element is removed and replaced):
    - 'preserveTargetChildren' specifies whether to copy the target element's children (true) or not (false).

- 'preserveTargetEvents' specifies whether to copy the target element's events (e.g. onmouseover) (true) or not (false).
- 'preserveTargetAttributes' specifies that all of the target element's attributes, none of them, or a list of specific attributes should be copied.

<parseXML>

Syntax:

<parseXML

id="*string*"

event="*string*"

string="*string*"

{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}

insertAs="{<u>child</u> | parent | sibling | replacement}"

offset="*signed integer*"

from="{<u>top</u> | bottom}"

preserveTargetChildren="{<u>true</u> | false}"

preserveTargetEvents="{<u>true</u> | false}"

preserveTargetAttributes="{<u>all</u> | none | *attr1;attr2;...attrN}*"

/>

Description:

Parses a string (text) containing valid XML data (a fragment or a full document) and from it, either inserts the fragment in the DOM or creates a full document.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'string' is the text containing the valid XML data.
- 'target' is the xpath pointing to wherever the fragment or document is to be placed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.

'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to place the new fragment or document. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.

- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to place the new fragment or document. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) or documentFragment in which to place the new fragment or document. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored. Note that if the XML data being loaded is of a different grammar than the current document, inserting it into the current document could cause problems, depending on what the XML Viewer allows. It is safer to provide a 'docID', which will be the ID of the documentFragment used to store this XML data. The data can then be referenced by any other behavior element 22, such as <setAttribute>, as described above.
- 'elementID' specifies the 'id' attribute of the element at which to insert the new fragment. If 'target' is provided, this attribute is ignored. If an entire document is being loaded, this attribute will be ignored.
- 'insertAs' specifies whether the new fragment is to be inserted as a child of the target element (the default), as the parent of the target element or as a sibling of the target element. If an entire document is being loaded or if 'docID' exists but 'elementID' does not, 'insertAs' is set to "replacement", which causes any document or documentFragment with an ID of 'docID' to be replaced with the new document or, if no such document exists, causes a new documentFragment to be created with an ID of 'docID'.
- If inserting as a child:
    - 'offset' specifies the number of nodes (not including comment nodes) from the top or bottom at which to insert the new fragment. A negative value specifies up towards the first child. A positive value specifies down towards the last child. If there are fewer nodes than specified by 'offset',

the fragment will be inserted as either the first child or the last child. If an entire document is being loaded, or if 'docID' exists but 'elementID' does not, this attribute is ignored.

- 'from' specifies whether 'offset' is relative to the top (first child) or bottom (last child). If an entire document is being loaded, or if 'docID' exists but 'elementID' does not, this attribute is ignored.
- 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.

- If inserting as the parent:
    - 'offset' is ignored.
    - 'from' is ignored.
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a sibling:
    - 'offset' specifies the number of nodes (not including comment nodes) before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert the new element. If there are fewer nodes than specified by 'offset', the element will be inserted as either the first child or the last child of the parent. If an entire document is being loaded, or if 'docID' exists but 'elementID' does not, this attribute is ignored.
    - 'from' is ignored.
    - 'preserveTargetChildren', 'preserveTargetEvents' and 'preserveTargetAttributes' are ignored.
- If inserting as a replacement (to an element, not to a document or documentFragment), in which the target element is removed and replaced:
    - 'preserveTargetChildren' specifies whether to copy the target element's children (true) or not (false).
    - 'preserveTargetEvents' specifies whether to copy the target element's events (e.g., onmouseover) (true) or not (false).
    - 'preserveTargetAttributes' specifies that all of the target element's attributes, none of them, or a list of specific attributes should be copied.

<printXML>

Syntax:

```
<printXML
        id="string"
        event="string"
        {source="xpath" | frameID="string" objectID="string" docID="string"
elementID="string"}
        {target="xpath" | xlink:href="URL"}
        output="string"
        {ignore="xpath" | ignoreMatch="string"}
/>
```

Description:

Converts the target document or element to text format and stores it within the 'output' attribute or as a file or at the desired location within an existing file.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'source' is the xpath pointing to the element, document or documentFragment to be converted to text. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element, document or documentFragment to be converted to text. If not provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element, document or documentFragment to be converted to text. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.
- 'docID' specifies either the document or documentFragment (e.g., an SVG or XHTML document) to be converted to text or specifies the 'id' attribute of the

document or documentFragment in which to find the element to be converted to text. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.

- 'elementID' specifies the 'id' attribute of the element to be converted to text. If 'source' is provided, this attribute is ignored.
- 'target' is the xpath pointing to the document or documentFragment or element within them, in which to store the text. If provided, 'xlink:href' is ignored.
- 'xlink:href' specifies the URL to a document or documentFragment or an element within them, in an external file. If 'target' is provided, this attribute is ignored.
- 'output' is the attribute in which the XML text is stored, if neither 'target' nor 'xlink:href' is provided. Actually providing a value for 'output' is meaningless and will result in the 'output' attribute immediately being set to an empty string anyway, to avoid confusion. This attribute is only used for storage of this action's resulting text, and thus is intended to be referenced by another element, such as <setAttribute>. Referencing attributes is described above.
- 'ignore' is an xpath that specifies a list of elements to ignore when outputing to text.
- 'ignoreMatch' specifies any combination of elements that match a particular type or have particular attributes, style properties or classes. For more precise matchings, parentheses can be used as well as the AND, OR and NOT operators. For example, to ignore all elements which use the CSS classes "info" or "warning" plus all elements which use the CSS classes "text1" and "fill2" plus all circles with a radius of 10 or 20 plus all invisible rectangles and circles that are not red or green, you could specify ignore="class('info' OR 'warning') OR class('text1' AND 'fill2') OR (element(circle) AND attribute('r='10';r='20';)) OR (element(rect;circle) AND style((visibility:hidden OR opacity:0 OR display:none) NOT (fill:red OR fill:green))". For more complicated matchings, JavaScript's syntax for regular expressions can also be used.

<removeAttribute>

Syntax:

```
<removeAttribute
        id="string"
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"}
        name="string"
        ns="string"
/>
```

Description:

Removes the specified attribute from the target element, which is different from setting the attribute to an empty string (""). Equivalent to <setAttribute> with the modify="remove" attribute.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose attribute is to be removed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose attribute is to be removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose attribute is to be removed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose attribute is to be removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element whose attribute is to be removed. If 'target' is provided, this attribute is ignored.

- 'name' specifies the name of the attribute to be removed.
- 'ns' specifies the namespace of the attribute to be removed. The namespace is prefixed to the attribute name, separated by a colon.

<removeClass>

Syntax:

<removeClass

id="*string*"

event="*string*"

{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}

cssName="*string*"

/>

Description:

Removes the specified CSS rule from the 'class' attribute for the target element. Equivalent to <setClass> with the modify="remove" attribute.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose 'class' attribute is to have the CSS rule removed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose 'class' attribute is to have the CSS rule removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose 'class' attribute is to have the CSS rule removed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.

- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose 'class' attribute is to have the CSS rule removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element whose 'class' attribute is to have the CSS rule removed. If 'target' is provided, this attribute is ignored.
- 'cssName' specifies the name of the CSS rule to remove from the element's 'class' attribute.

<removeData>

Syntax:

<removeData
    id="*string*"
    event="*string*"
    {target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*"
    elementID="*string*"}
    offset="*signed integer*"
    from="{top | bottom}"
    count="*integer*"
    substring="*string*"
    occurrence="*string*"
/>

Description:

Removes all data or the specified data from the target element. Equivalent to <setData> with the modify="remove" attribute.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose data is to be removed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.

- ‘frameID’ specifies the ‘id’ attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose data is to be removed. If not provided, the current frame is assumed. If ‘target’ is provided, this attribute is ignored.
- ‘objectID’ specifies the ‘id’ attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose data is to be removed. If not provided, the current object is assumed. If ‘target’ is provided, this attribute is ignored.
- ‘docID’ specifies the ‘id’ attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose data is to be removed. If not provided, the current document is assumed. If ‘target’ is provided, this attribute is ignored.
- ‘elementID’ specifies the ‘id’ attribute of the element whose data is to be removed. If ‘target’ is provided, this attribute is ignored.
- ‘offset’ specifies the number of characters from the beginning or end of the data (as specified by the ‘from’ attribute) at which to remove data. If ‘count’ is not provided, this attribute is ignored. The default is 0.
- ‘from’ specifies whether the ‘offset’ attribute is relative to the beginning (top), which is the default, or end (bottom) of the data.
- ‘count’ specifies the number of consecutive characters after the ‘offset’ that is to be removed.
- ‘substring’ specifies text to search for in the element’s data, which, if found, will be removed.
- ‘occurrence’ specifies which occurrence of ‘substring’ should be removed. The default is 1, signifying the first occurrence. If ‘substring’ is not provided, this attribute is ignored.

<removeDocument>

Syntax:

<removeDocument

id="*string*"

```
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"}
/>
```

Description:

Removes the specified document or documentFragment.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the document or documentFragment to be removed. If provided, 'frameID', 'objectID' and 'docID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the document or documentFragment to be removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the document or documentFragment to be removed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document or documentFragment (e.g., an SVG or XHTML document) to be removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.

<removeElement>

Syntax:

```
<removeElement
        id="string"
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"
        elementID="string"}
/>
```

Description:

Removes a specified element from the DOM.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element to be removed. If provided, 'frameID', 'objectID' and 'docID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element to be removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element to be removed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document or documentFragment (e.g., an SVG or XHTML document) in which to find the element to be removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element to be removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.

<removeLink>

Syntax:

```
<removeLink
        id="string"
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"
        elementID="string"}
/>
```

Description:

Removes the associated link for the target element, in effect removing the <a xlink:href=""></a> that surrounds the target element. Equivalent to <setLink> with the modify="remove" attribute.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element to have its associated link removed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element to have its associated link removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element to have its associated link removed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element to have its associated link removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element to have its associated link removed. If 'target' is provided, this attribute is ignored.

<removeProcessingInstruction>

Syntax:

```
<removeProcessingInstruction
        id="string"
        event="string"
        data="string"
/>
```

Description:

Removes a processingInstruction from the document, e.g., <? xml version="1.0"?>

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'data' specifies the processingInstruction to be removed, e.g., xml version="1.0".

<removeStyle>

Syntax:

```
<removeStyle
        id="string"
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"
        elementID="string"}
        {name="string" | entityName="string"}
/>
```

Description:

Removes the specified style property and/or entity from the 'style' attribute for the target element. Equivalent to <setStyle> with the modify="remove" attribute.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose 'style' attribute is to have the specified property removed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose 'style' attribute is to have the specified property removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose 'style' attribute is to be set.

If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.

- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose 'style' attribute is to have the specified property removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element whose 'style' attribute is to have the specified property removed. If 'target' is provided, this attribute is ignored.
- 'name' specifies the name of the style property to be removed from the element's 'style' attribute. e.g., stroke-width, stroke, fill, font-size, etc.
- 'entityName' specifies the name of the entity to be removed from the element's 'style' attribute. If 'name' is provided, then both the 'name' and 'entityName' will be removed.

<removeStyleSheet>

Syntax:

<removeStyleSheet
 id="*string*"
 event="*string*"
 cssName="*string*"
/>

Description:

Removes the specified CSS rule from the CDATA block of a <style> element Equivalent to <setStyleSheet> with the modify="remove" attribute.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'cssName' specifies the reference name of the CSS rule, e.g., ".stro" or "text.info"

<setAttribute>

Syntax:

```
<setAttribute
        id="string"
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"
        elementID="string"}
        name="string"
        ns="string"
        modify="{append | precede | remove | replace}"
        delimiter="string"
        value="string"
        savePreviousValue="{true | false"}
/>
```

Description:

Creates, modifies, replaces or removes an attribute for the target element.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose attribute is to be set. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose attribute is to be set. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose attribute is to be set. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g. an SVG or XHTML document) in which to find the element whose attribute is to be set. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.

- 'elementID' specifies the 'id' attribute of the element whose attribute is to be set. If 'target' is provided, this attribute is ignored.
- 'name' specifies the name of the attribute to be set.
- 'ns' specifies the namespace of the attribute to be set. The namespace is prefixed to the attribute name, separated by a colon. If the attribute is being created, and the namespace is not defined in the document's root element, it will be defined in the target element.
- 'modify' specifies whether the attribute is to be created or replaced (replace), removed (remove) or modified, with the new text added to the beginning (precede) or the end (append) of the existing text.
- 'delimiter' specifies the text that is to separate the previous value from the new value. If 'modify' is not 'append' or 'precede', this attribute is ignored.
- 'value' specifies the value that the attribute is to be given. If modify="remove", this attribute is ignored.
- 'savePreviousValue' specifies whether to save the previous value of the attribute (true) or not (false). If saved, the previous value will be stored in a new attribute with the same name prefixed with "previous_" for the target element.

<setClass>

Syntax:

<setClass
    id="*string*"
    event="*string*"
    {target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}
    cssName="*string*"
    modify="{append | precede | remove | <u>replace</u>}"
    savePrevious="{true | <u>false</u>}"
/>

Description:

Adds or removes a CSS rule in the 'class' attribute for the target element, or replaces the contents of the 'class' attribute entirely. If adding or replacing, the 'class' attribute will be created, if not already there.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose 'class' attribute is to be set. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose 'class' attribute is to be set. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose 'class' attribute is to be set. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose 'class' attribute is to be set. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element whose 'class' attribute is to be set. If 'target' is provided, this attribute is ignored.
- 'cssName' specifies the name of the CSS rule (contained within a CDATA block, as a child of a <style type="text/css"> element) to replace, be added to, or be removed from the element's 'class' attribute
- 'modify' specifies whether the CSS rule should be inserted at the beginning (precede), inserted at the end (append), or removed from (remove) the existing 'class' value, or whether it should replace the contents of the 'class' attribute entirely. If adding or replacing, the 'class' attribute will be created, if not already

there. If adding ("append" or "precede"), a whitespace character is used for the delimiter.

- 'savePreviousValue' specifies whether to save the previous value of the 'class' attribute (true) or not (false). If saved, the previous value will be stored in the 'previous_class' attribute for the target element.

<setComment>

Syntax:

<setComment
id="*string*"
event="*string*"
{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}
offset="*signed integer*"
value="*string*"
modify="{append | create | precede | remove | replace}"
/>

Description:

Creates, removes or modifies a comment (i.e., <!--comment-->) in the desired location.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location at which the comment is to be created, removed or modified. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the reference element. If not provided, the current frame is assumed. If 'source' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the document (e.g., an HTML <object> or <embed> element) in which to find the reference element. If not provided, the current object is assumed. If 'source' is provided, this attribute is ignored.

- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the reference element. If not provided, the current document is assumed. If 'source' is provided, this attribute is ignored.
- 'elementID' specifies the reference element. If 'source' is provided, this attribute is ignored.
- 'offset' specifies the number of comment nodes before (if 'offset' is negative) or after (if 'offset' is positive) the target element at which to insert, remove or modify a comment. If there are fewer consecutive comment nodes than specified by 'offset', the first or last comment node will be created/removed/modified.
- 'value' specifies the comment's actual text.
- 'modify' specifies whether a new comment is to be inserted before the target (create), or whether the existing target comment node is to be removed (remove), replaced entirely (replace) or modified, with the new text added to the beginning (precede) or the end (append) of the existing text.

<setData>

Syntax:

<setData
    id="*string*"
    event="*string*"
    {target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*"
    elementID="*string*"}
    value="*string*"
    modify="{append | precede | remove | <u>replace}</u>"
    offset="*signed integer*"
    from="{<u>top</u> | bottom}"
    count="*integer*"
    substring="*string*"
    occurrence="*string*"
    savePrevious="{true | <u>false}</u>"
/>

Description:

Creates, modifies, replaces or removes data for the target element.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose data is to be set. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose data is to be set. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose data is to be set. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose data is to be set. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element whose data is to be set. If 'target' is provided, this attribute is ignored.
- 'value' specifies the string to replace or add to element's data. If modify="remove", this attribute is ignored.
- 'modify' specifies whether the data is to be created or replaced (replace), removed (remove) or modified, with the new text added to the beginning (precede) or the end (append) of the existing text or substring.
- 'offset' specifies the number of characters from the beginning or end of the data (as specified by the 'from' attribute) at which to insert the new data. If 'modify' does not equal "append" or "precede", this attribute is ignored.
- 'from' specifies whether the 'offset' attribute is relative to the beginning (top) or end (bottom) of the data.

- 'count' specifies the number of consecutive characters after the 'offset' to be replaced by the new data or to have the new data appended after. If 'modify' does not equal "replace" or "append", this attribute is ignored.
- 'substring' specifies text to search for in the element's data. If found, it may be removed or replaced or the new data may be inserted at the beginning or end of it, depending on the 'modify' attribute.
- 'occurrence' specifies which occurrence of 'substring' should be removed, replace or modified. The default is 1, signifying the first occurrence. If 'substring' is not provided, this attribute is ignored.
- 'savePreviousValue' specifies whether to save the previous data (true) or not (false). If saved, the previous data will be stored in a new 'previousData' attribute for the target element.

<setEntity>

Syntax:

<setEntity

id="*string*"

event="*string*"

name="*string*"

value="*string*"

modify="{append | remove | <u>replace</u>}"

/>

Description:

Creates, modifies or removes an entity, e.g., <!ENTITY st0 "fill:none;stroke:black;">.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'name' specifies the reference name of the entity, e.g., "st0"
- 'value' specifies the string that the entity resolves to, e.g., "fill:none;stroke:black;". This is ignored if 'modify' is set to 'remove'.

- 'modify' specifies whether to append to the entity's current value, replace the entity's value, or remove the entity altogether. If 'modify' is set to 'remove', then 'value' is ignored.

```
<setEventListener
        id="string"
        event="string"
        eventName="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"
elementID="string"}
        handlerID="string"
        handlerFunction="string"
        modify="{add | remove | replace}"
/>
```

Description:

Sets an event listener on the desired element. This does not add an event attribute (e.g., onmouseover) to the element.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'eventName' specifies the name of the event that the element should listen for (e.g., mouseover).
- 'target' is the xpath pointing to the location of the element to set an event listener on. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element to set an event listener on. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element to set an event listener on. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.

- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element to set an event listener on. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element to set an event listener on. If 'target' is provided, this attribute is ignored.
- 'handlerID' specifies the 'id' attribute of the behaviour element or <action> container for behaviours to handle the event. If 'handlerFunction' is also defined, it will handle the event after the action element is done.
- 'handlerFunction' specifies the JavaScript function name to handle the event. If 'handlerID' is also defined, it will handle the event first before passing control on to the JavaScript function.
- 'modify' specifies whether to remove, replace or add to any existing event listener(s).

<setLink>

Syntax:

<setLink

id="*string*"

event="*string*"

{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}

xlink:href="*URL*"

modify="{remove | <u>set</u>}"

savePreviousValue="{true | <u>false</u>"}

/>

Description:

Sets or removes a link for the target element, in effect creating or removing an <a xlink:href=""></a> around the target element.

Attributes:

- 'id' allows this action element to be referenced later.

- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element to have an associated link set or removed. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element to have an associated link set or removed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose attribute is to have an associated link set or removed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element to have an associated link set or removed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element to have an associated link set or removed. If 'target' is provided, this attribute is ignored.
- 'xlink:href' specifies the URL (the link) to be associated with the target element. If modify="remove", this attribute is ignored.
- 'modify' specifies whether the attribute is to be created or replaced (replace), removed (remove) or modified, with the new text added to the beginning (precede) or the end (append) of the existing text.
- 'savePreviousValue' specifies whether to save the previous value of the link (true) or not (false). If saved, the previous value will be stored in the attribute 'previousLink' for the *target element*, not for the <a> element, since <setLink> effectively abstracts the author from needing to understand how links really work.

<setStyle>

Syntax:

<setStyle

id="*string*"
event="*string*"
{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*"
elementID="*string*"}
{name="*string*" value="*string*" | entityName="*string*"}
modify="{append | precede | remove | <u>replace</u>"}
savePrevious="{true | <u>false</u>}"
/>

Description:

Adds or removes a style property or entity in the 'style' attribute for the target element, or replaces the contents of the 'class' attribute entirely. If adding or replacing, the 'style' attribute will be created, if not already there.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose 'style' attribute is to be set. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose 'style' attribute is to be set. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose 'style' attribute is to be set. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose 'style' attribute is to be set. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.

- 'elementID' specifies the 'id' attribute of the element whose 'style' attribute is to be set. If 'target' is provided, this attribute is ignored.
- 'name' specifies the name of the style property to be added, removed or replaced. e.g., stroke-width, stroke, fill, font-size, etc.
- 'value' specifies the value of the style property to be added, removed or replaced.
- 'entityName' specifies the name of the entity to be added to, to be removed from or to replace the element's 'style' attribute. If 'name' is provided, then this attribute is ignored.
- If 'name' is provided, then
    - 'modify' specifies that the existing 'style' attribute's value should have 'value' inserted at the beginning (precede) or the end (append), delimited by a semicolon character, or that the existing style property (e.g., 'stroke-width'), if there is one, should be removed (remove) or replaced with 'value' (replace).
- If 'entityName' is provided and 'name' is not provided, then
    - 'modify' specifies whether the entity should replace the existing 'style' value (replace), which creates it if the 'style' attribute does not exist, be removed from the existing 'style' value (remove), be appended to the existing 'style' value (append) or be inserted at the front of the existing 'style' value (precede). If "append" or "precede", a semicolon character (;) is used for the delimiter.
- 'savePreviousValue' specifies whether to save the previous value of the 'style' attribute (true) or not (false). If saved, the previous value will be stored in the 'previous_style' attribute for the target element.

<setStyleSheet>

Syntax:

<setStyleSheet

id="*string*"

event="*string*"

cssName="*string*"

```
        value="*string*"
        modify="{append | precede | remove | replace"}
/>
```

Description:

Creates, modifies or removes a CSS rule within the CDATA block of a <style> element, e.g.,

```
    <style type="text/css">
      <![CDATA[ .str0    { stroke:red;stroke-width:2 }
            text.info  { color:green }
            text.error { color:red } ]]>
    </style>
```

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'cssName' specifies the reference name of the CSS rule, e.g., ".stro" or "text.info"
- 'value' specifies the string that the CSS rule resolves to, e.g., "stroke:red;stroke-width:2". This is ignored if 'modify' is set to 'remove'.
- 'modify' specifies whether to append to the rule's current value, replace the rule's value, or remove the rule altogether. If 'modify' is set to 'remove', then 'value' is ignored.

<viewDocument>

Syntax:

```
<viewDocument
        id="string"
        event="string"
        {target="xpath" | frameID="string" objectID="string" docID="string"}
/>
```

Description:

Causes the viewer to view the target document, while still preserving the current document.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the document to be viewed.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the document to be viewed. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the document to be viewed. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) to be viewed. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.

SVG DOM:

<setTransform>

Syntax:

<setTransform

id="*string*"

event="*string*"

{target="*xpath*" | frameID="*string*" objectID="*string*" docID="*string*" elementID="*string*"}

{matrix="*string*" | scale="*string*" | scaleX="*string*" scaleY="*string*" | translateX="*string*" translateY="*string*" | rotate="*string*" cx="*string*" cy="*string*" | skewX="*string*" | skewY="*string*"}"

absolute={true | <u>false</u>}

hAlign={left | middle | right | integer}

vAlign={top | middle | bottom | integer}

referenceID="*string*"

savePrevious="{true | <u>false</u>}"

/>

Description:

Creates, modifies, replaces or removes the 'transform' attribute for the target element. At markup level, this has the effect of replacing the'transform' attribute's value with a single matrix transformation. The SVG version 1.0 specification provides detailed information regarding SVG transformations.

Attributes:

- 'id' allows this action element to be referenced later.
- 'event' specifies the event that triggers this action.
- 'target' is the xpath pointing to the location of the element whose 'transform' attribute is to be set. If provided, 'frameID', 'objectID', 'docID' and 'elementID' are ignored.
- 'frameID' specifies the 'id' attribute of the frame (e.g., an HTML <frame> element) in which to find the element whose 'transform' attribute is to be set. If not provided, the current frame is assumed. If 'target' is provided, this attribute is ignored.
- 'objectID' specifies the 'id' attribute of the object (e.g., an HTML <object> or <embed> element) in which to find the element whose 'transform' attribute is to be set. If not provided, the current object is assumed. If 'target' is provided, this attribute is ignored.
- 'docID' specifies the 'id' attribute of the document (e.g., an SVG or XHTML document) in which to find the element whose 'transform' attribute is to be set. If not provided, the current document is assumed. If 'target' is provided, this attribute is ignored.
- 'elementID' specifies the 'id' attribute of the element whose 'transform' attribute is to be set. If 'target' is provided, this attribute is ignored.
- 'matrix' specifies a matrix transformation to be applied to the target element. The matrix is of the form "a b c d e f", where a, b, c, d, e and f are coefficients of the 3x3 transformation matrix (as in the SVG version 1.0 specification).
- 'scale' specifies a scale factor to be applied to the target element along both the x- and y-axis.

- 'scaleX' specifies a scale factor to be applied to the target element along the x-axis.
- 'scaleY' specifies a scale factor to be applied to the target element along the y-axis.
- 'translateX' specifies a translation to be applied to the target element along the x-axis.
- 'translateY' specifies a translation to be applied to the target element along the y-axis.
- 'rotate' specifies a rotation, in degrees about a given point, to be applied to the target element.
- 'cx' specifies the x-coordinate of the point about which to rotate the element. If either 'rotate' or 'cy' is not defined, this attribute is ignored.
- 'cy' specifies the y-coordinate of the point about which to rotate the element. If either 'rotate' or 'cx' is not defined, this attribute is ignored.
- 'skewX' specifies a skew, in degrees, along the x-axis.
- 'skewY' specifies a skew, in degrees, along the y-axis.
- 'absolute' specifies whether the new transformation should be applied to the element's current transformation (false) or should replace the element's current transformation (true).
- 'hAlign' specifies that a translation should be automatically calculated and applied to the target element so that after the transformation, anything at the coordinates occupied by the target element's left edge, middle, or right edge will have its pre-transformed position preserved. If 'referenceID' is supplied, then the element with that ID will have its position preserved, as specified by the 'hAlign' attribute, rather than the target element. If hAlign equals an integer, that integer represents the pre-transformed x-coordinate of the position that you wish to be preserved after the transformation.
- 'vAlign' specifies that a translation should be automatically calculated and applied to the target element so that after the transformation, anything at the coordinates occupied by the target element's top edge, middle, or bottom edge will have its pre-transformed position preserved. If 'referenceID' is supplied, then the element

with that ID will have its position preserved, as specified by the 'vAlign' attribute, rather than the target element. If vAlign equals an integer, that integer represents the pre-transformed y-coordinate of the position that you wish to be preserved after the transformation.

- 'referenceID' specifies the 'id' attribute of the element whose pre-transformed position (specified by the 'hAlign' and 'vAlign' attributes) is to be preserved after the transformation. This element does not actually get a transformation applied to it, it is merely used to define the coordinates that should be preserved after the transformation. For example, there may be two elements contained within a group and you want to transform both of them, but always preserve the centre of just one of them. The target would be the <g> container element and referenceID would be the ID of the element whose position is to be preserved.
- 'savePreviousValue' specifies whether to save the previous value of the 'transform' attribute (true) or not (false). If saved, the previous value will be stored in the 'previous_transform' attribute for the target element.

WHAT IS CLAIMED IS:

1. A system for manipulating a document object model, the system comprising:

a collection of document object model behavior elements, each behavior element comprising:

a name following a predetermined naming convention;

an event attribute for associating the behavior element to an event; and

other attributes for describing features of the behavior element;

a collection of scripts for performing actions associated with the set of behavior elements, each script associated with a behavior element; and

an initialization function for directing the processing of one or more behavior elements in a document object model.

2. The system as claimed in claim 1, wherein the predetermined naming convention comprises having a constant prefix to the name of the element.

3. The system as claimed in claim 1, wherein the event attribute comprises an attribute which is set to a specific event.

4. The system as claimed in claim 3, wherein the behavior element is dormant until the specific event occurs.

5. The system as claimed in claim 3, wherein once the specific event occurs, the behavior element performs the actions.

6. The system as claimed in claim 3, wherein the specific event is the loading of a document object model into a viewer.

7. The system as claimed in claim 3, wherein the specific event is the clicking of an object in a browser window, the object associated with a behavior element

8. The system as claimed in claim 3, wherein the specific event is a cursor associated with a mouse of a computer system passing over an object in a browser window, the object associated with a behavior element.

9. The system as claimed in claim 1, wherein the behavior element is associated with an extensible markup language element.

10. The system as claimed in claim 9, wherein the behavior element is a child of the extensible markup language element.

11. The system as claimed in claim 9, wherein the behavior element is a parent of the extensible markup language element.

12. The system as claimed in claim 1, wherein the actions comprise behavioral mutations of an output of extensible markup language elements.

13. The system as claimed in claim 1, wherein the initialization function contains instructions for traversing each node in the document object model and for searching and calling functions associated with behavior elements having names following the predetermined naming convention.

14. The system as claimed in claim 1, further comprising:

a collection of behavior attributes for adding to existing regular extensible markup language elements in a document object model, the behavior attributesfollowing the predetermined naming convention; and

a collection of scripts for performing actions associated with the collection of behavior attributes, each script associated with a behavior attribute.

15. The system as claimed in claim 14, wherein the initialization function contains instructions for traversing each node in the document object model and for searching and

calling functions associated with behavior elements and behavior attributes having names following the predetermined naming convention.

16. The system as claimed in claim 1, wherein the collection of scripts is stored in a memory location of a computer system.

17. The system as claimed in claim 1, wherein each script in the collection of scripts is stored in a separate file.

18. The system as claimed in claim 1, wherein the collection of behavior elements comprises a markup language.

19. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:createElement behavior element for creating a new element and inserting the newly created element in a desired location in the document object model, the dsvg:createElement behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a new element identification attribute for specifying the value of an identification attribute of the newly created element;

an element name attribute for specifying the name of the element to be created;

an attributes attribute for containing all of the attributes for the newly created element;

a namespace attribute for specifying the namespace of the newly created element;

a target attribute for specifying the xpath pointing to the location at which the new element is to be inserted;

an insert as attribute for specifying whether the new element is to be inserted as a child, parent or sibling of a target element;

an offset attribute for specifying the number of nodes before or after the target element;

a from attribute for specifying whether the offset attribute is relative to first child or last child of the target element;

a preserve target children attribute for specifying whether to copy the children of the target element;

a preserve target events attribute for specifying whether to copy the events of the target element; and

a preserve target attributes attribute for specifying the attributes of the target element.

20. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:createEvent behavior element for creating an event and dispatching the event to a desired target in the document object model, the dsvg:createEvent behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

an event name attribute for specifying the type of the event;

a source attribute for specifying the xpath to the element that the target will believe created the event;

a target attribute for specifying the xpath to the element to which the event is dispatched;

an event phase attribute for specifying a phase of event flow that is currently being evaluated;

a bubbles attribute for specifying whether or not the event can bubble;

a cancelable attribute for specifying whether or not the event can have its default actions prevented;

a time stamp attribute for specifying the time at which the event was created;

a stop propagation attribute for preventing further propagation of an event during event flow; and

a prevent default attribute for specifying whether or not the event is to be cancelled, so that any default action normally taken by an implementation as a result of the event will not occur.

21. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:loadXML behavior element for creating a new element and inserting the newly created element in a desired location in the document object model, the dsvg:loadXML behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a source attribute for specifying an xpath to a document or element within a document in an external file;

a target attribute for specifying an xpath pointing to where the document or fragment is to be placed;

an insert as attribute for specifying whether the new fragment is to be inserted as a child, a parent or a sibling of the target element;

an offset attribute for specifying the number of nodes before or after the target element at which to insert the new element;

a from attribute for specifying whether the offset attribute is relative to the first child or last child of the target element;

a preserve target children attribute for specifying whether to copy the children of the target element;

a preserve target events attribute for specifying whether to copy the events of the target element; and

a preserve target attributes attribute for specifying the attributes of the target element.

22. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:setAttribute behavior element for creating, modifying, replacing or removing an attribute for a target element in the document object model, the dsvg:setAttribute behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a target attribute for specifying an xpath pointing to a location of the target element;

a name attribute for specifying the name of the attribute to be set;

a namespace attribute for specifying a namespace of the attribute to be set;

a modify attribute for specifying whether the attribute is to be created, replaced, removed or modified, with new text added to the beginning or the end of existing text;

a delimiter attribute for specifying text that is to separate a previous value from a new value;

a value attribute for specifying a value that the attribute is to be given; and

a save previous value attribute for specifying whether to save the previous value of the attribute.

23. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:setClass behavior element for modifying contents of a class attribute of a target element in the document object model, the dsvg:setClass behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a target element for specifying an xpath pointing to a location of the target element;

an element identification attribute for specifying the identification attribute of the target element;

a css name attribute for specifying the name of a CSS rule to replace, be added to, or be removed from the class attribute of the target element;

a modify attribute for specifying how the CSS rule should modify the class attribute of the target element; and

a save previous value attribute for specifying whether to save the previous value of the class attribute of the target element.

24. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:setData behavior element for modifying data of a target element in the document object model, the dsvg:setData behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a target attribute for specifying an xpath pointing to a location of the target element;

an element identification attribute for specifying the identification attribute of the target element;

a value attribute for specifying the string to replace or add to data of the target element;

a modify attribute for specifying how previous data it to be modified by new data;

an offset attribute for specifying the number of characters from the beginning or end of the data at which to insert new data;

a from attribute for specifying whether the offset attribute is relative to the beginning or end of the data;

a count attribute for specifying the number of consecutive characters after the offset attribute to be replaced by the new data or to have the new data appended after;

a substring attribute for specifying text to search for in the data of the target element;

an occurrence attribute for specifying which occurrence of the substring attribute should be removed, replaced or modified; and

a save previous value attribute for specifying whether to save the previous data of the target element.

25. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:setStyle behavior element for modifying the contents of a style attribute of a target element and for replacing the contents of a class attribute in a target element in the document object model, the dsvg:setStyle behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a target attribute for specifying an xpath pointing to a location of the target element;

a name attribute for specifying the name of the style property to be added, removed or replaced;

a value attribute for specifying the value of the style property to be added, removed or replaced;

a modify attribute for specifying how to modify the style attribute of the target element; and

a save previous value attribute for specifying whether to save the previous value of the style attribute of the target element.

26. The system as claimed in claim 1, wherein the collection of behavior elements comprises a dsvg:setTransform behavior element for modifying a transform attribute of a target element in the document object model, the dsvg:setTransform behavior element comprising:

a namespace following the predetermined naming convention;

an event attribute for specifying the event that triggers the behavior element, the event attribute settable to a string;

a collection of other attributes comprising:

an identification attribute for referencing the behavior element;

a target attribute for specifying an xpath pointing to the location of the target element;

a matrix attribute for specifying a matrix transformation to be applied to the target element;

an absolute attribute for specifying how to apply a new transformation with respect a current transformation of the target element;

an hAlign attribute for specifying how to calculate and apply a translation to the target element;

a vAlign attribute for specifying how to calculate and apply a translation to the target element;

a reference identification attribute for specifying the identification attribute of the target element; and

a save previous value attribute for specifying whether to save the previous value of the transform attribute of the target element.

27. A system for manipulating a document object model, the system comprising:

a collection of scripts for performing actions associated with markup behavior elements, each script associated with a behavior element; and

an initialization function for directing the processing of one or more behavior elements in a document object model.

28. A method of manipulating a document object model, the method comprising the steps of:

searching for a designated element in a document object model; and

calling a script associated with the designated element.

29. The method as claimed in claim 28, wherein the step of searching includes the steps of:

traversing each node in the document object model; and

determining whether an element has a name which follows a designated naming convention.

30. The method as claimed in claim 29, wherein the designated naming convention comprises appending a prefix to the name of the designated element.

31. The method as claimed in claim 28, wherein the step of calling a script includes the steps of:

dynamically generating a function name associated with the designated element;

passing an object associated with the designated element as a parameter of the generated function;

retrieving the attributes of the object; and

performing a function stored in memory having the generated function name.

32. The method as claimed in claim 31, wherein the step of dynamically generating includes the steps of:

determining if the name of the designated element contains a designated prefix;

generating a function name comprising of the name of the designated element;

assigning an object associated with the designated element as the parameter of the function; and

assigning predetermined instructions of the designated element as steps for the function to perform.

33. The method as claimed in claim 28, wherein the step of calling a script includes the steps of:

determining which script in a collection of scripts is associated with the designated element; and

calling the script.

34. The method as claimed in claim 28, further comprising the steps of:

searching for a designated attribute in an element in a document object model; and

calling a script associated with the designated attribute.

35. The method as claimed in claim 34, wherein the step of searching for a designated attribute comprises the steps of:

searching attributes of an element in a document object model;

determining whether an element attribute has a name which follows a designated naming convention.

36. The method as claimed in claim 35, wherein the naming convention comprises appending a prefix to the name of the designated attribute.

37. The method as claimed in claim 34, wherein the step of calling a script includes the steps of:

determining if the name of the designated attribute contains a designated prefix;

generating a function name comprising of the name of the designated attribute;

assigning an object associated with the designated attribute as the parameter of the function name ; and assigning predetermined instructions of the designated attribute as steps for a function having the function name to perform.

38. The method as claimed in claim 34, wherein the step of calling a script includes the steps of:

dynamically generating a function name associated with the designated attribute;

passing an object associated with the designated attribute as a parameter of the generated function name;

receiving the attributes of the object; and

performing a function stored in memory having the generated function name.

39. The method as claimed in claim 38, wherein the step of dynamically generating comprises the steps of:

determining if the name of the designated attribute contains a designated prefix;

generating a function name comprising of the name of the designated attribute;

assigning an object associated with the designated attribute as the parameter of the function; and

assigning predetermined instructions of the designated attribute as steps for the function to perform.

40. The method as claimed in claim 34, wherein the step of calling a script includes the steps of:

determining which script in a collection of scripts is associated with the designated attribute; and

calling the script.

41. A method of manipulating a document object model, the method comprising the steps of:

adding an event listener to an element having a designated element as a child in the document object model;

receiving an event which is equal to an event attribute setting in the designated element; and

calling a script associated with the designated element.

42. The method as claimed in claim 41, wherein the designated element has a name following a naming convention comprised of appending a prefix to the name of the designated attribute.

43. The method as claimed in claim 41, wherein the step of calling a script includes the steps of:

determining if the name of the designated element contains a designated prefix;

generating a function name comprising of the name of the designated element;

assigning an object associated with the designated element as the parameter of the function name; and

assigning predetermined instructions of the designated element as steps for a function having the function name to perform.

44. The method as claimed in claim 41, wherein the step of calling a script includes the steps of:

dynamically generating a function name associated with the designated element;

passing an object associated with the designated element as a parameter of the generated function name;

receiving the attributes of the object; and

performing a function stored in memory having the generated function name.

45. The method as claimed in claim 44, wherein the step of dynamically generating comprises the steps of:

determining if the name of the designated element contains a designated prefix;

generating a function name comprising of the name of the designated element;

assigning an object associated with the designated element as the parameter of the function; and assigning predetermined instructions of the designated element as steps for the function to perform.

46. The method as claimed in claim 41, wherein the step of calling a script includes the steps of:

determining which script in a collection of scripts is associated with the designated element; and

calling the script.

47. A method of creating an element for manipulating a document object model, the method comprising the steps of:

categorizing low level actions into behavior groupings;

determining common attributes of a behavior grouping; and

creating a behavior element having the common attributes of the behavior grouping.

48. The method as claimed in claim 47, wherein a plurality of behavior elements are created.

49. The method as claimed in claim 47, wherein the step of categorizing includes the steps of:

organizing the low level actions into groups of similar actions; and

designating behavior names to the groupings.

50. The method as claimed in claim 47, wherein the step of determining includes the steps of:

analyzing variations of the high level actions; and

compiling a list of attributes needed to perform the variations.

51. The method as claimed in claim 47, wherein the step of creating includes the steps of:

assigning a name to the behavior element pursuant to a predetermined naming convention; and

assigning the common attributes to the behavior element.

52. The method as claimed in claim 47, further comprising the step of initiating default settings for the behavior element.

53. The method as claimed in claim 47, further comprising the step of saving the behavior element is an independent file.

54. Computer readable media storing the instructions and/or statements for use in the execution in a computer of a method of manipulating a document object model, the method comprising steps of:

searching for a designated element in a document object model; and

calling a script associated with the designated element.

55. Electronic signals for use in the execution in a computer of a method of manipulating a document object model, the method comprising steps of:

searching for a designated element in a document object model; and

calling a script associated with the designated element.

56. A computer program product for use in the execution in a computer of a method of manipulating a document object model, the computer program product comprising:

a collection of scripts for performing actions associated with markup behavior elements, each script associated with a behavior element; and

an initialization function for directing the processing of one or more behavior elements in a document object model.

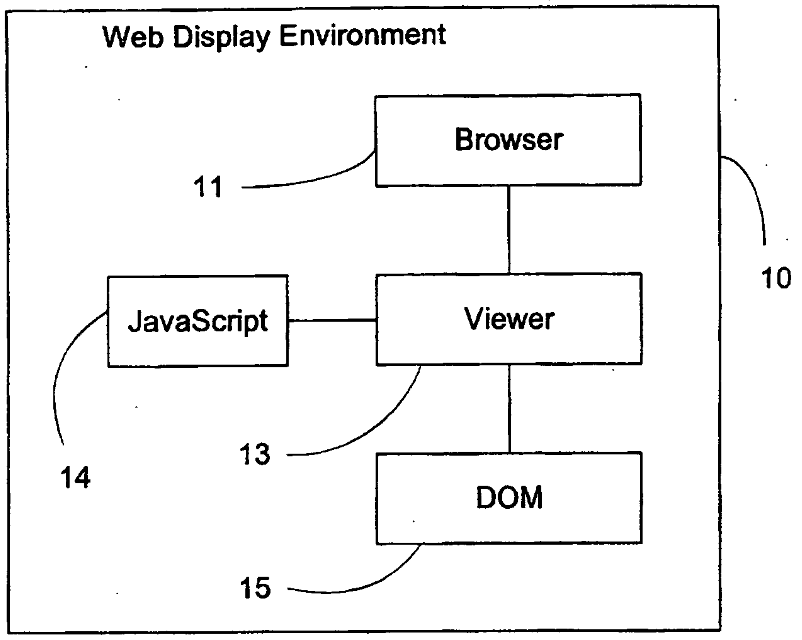

Figure 1
PRIOR ART

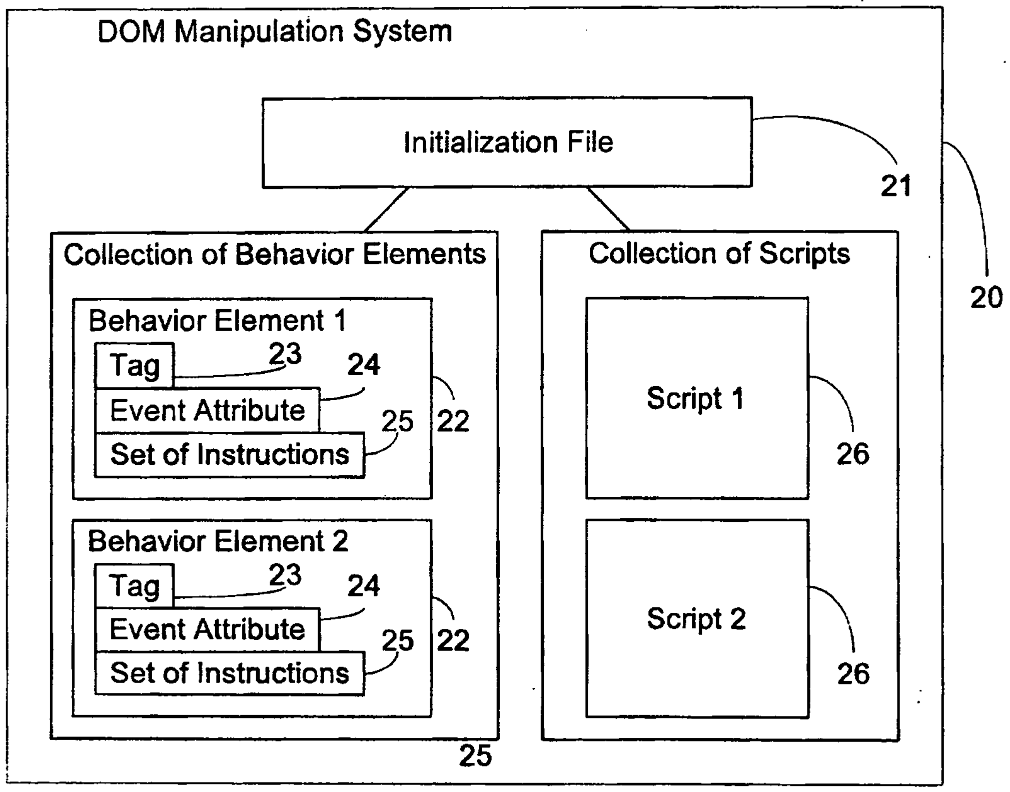

Figure 2

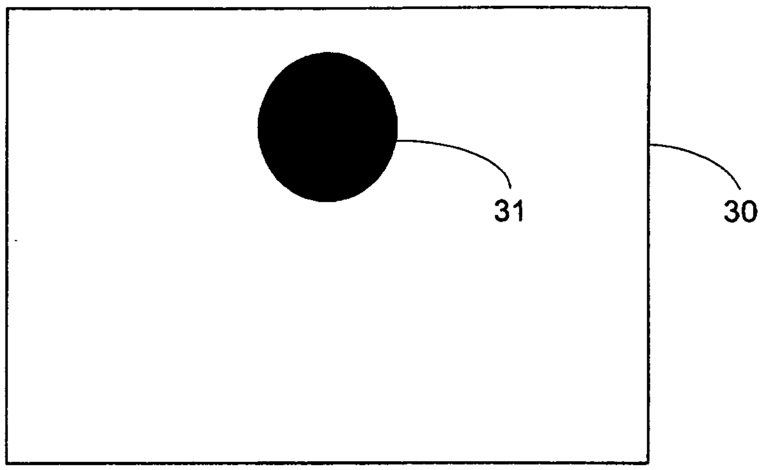

Figure 3

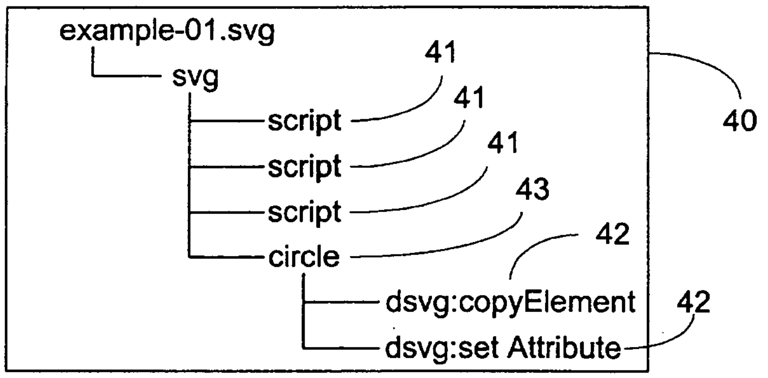

Figure 4

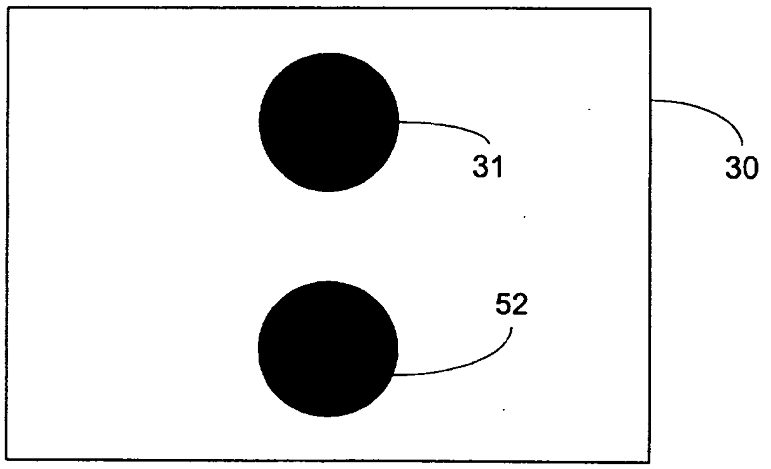

Figure 5

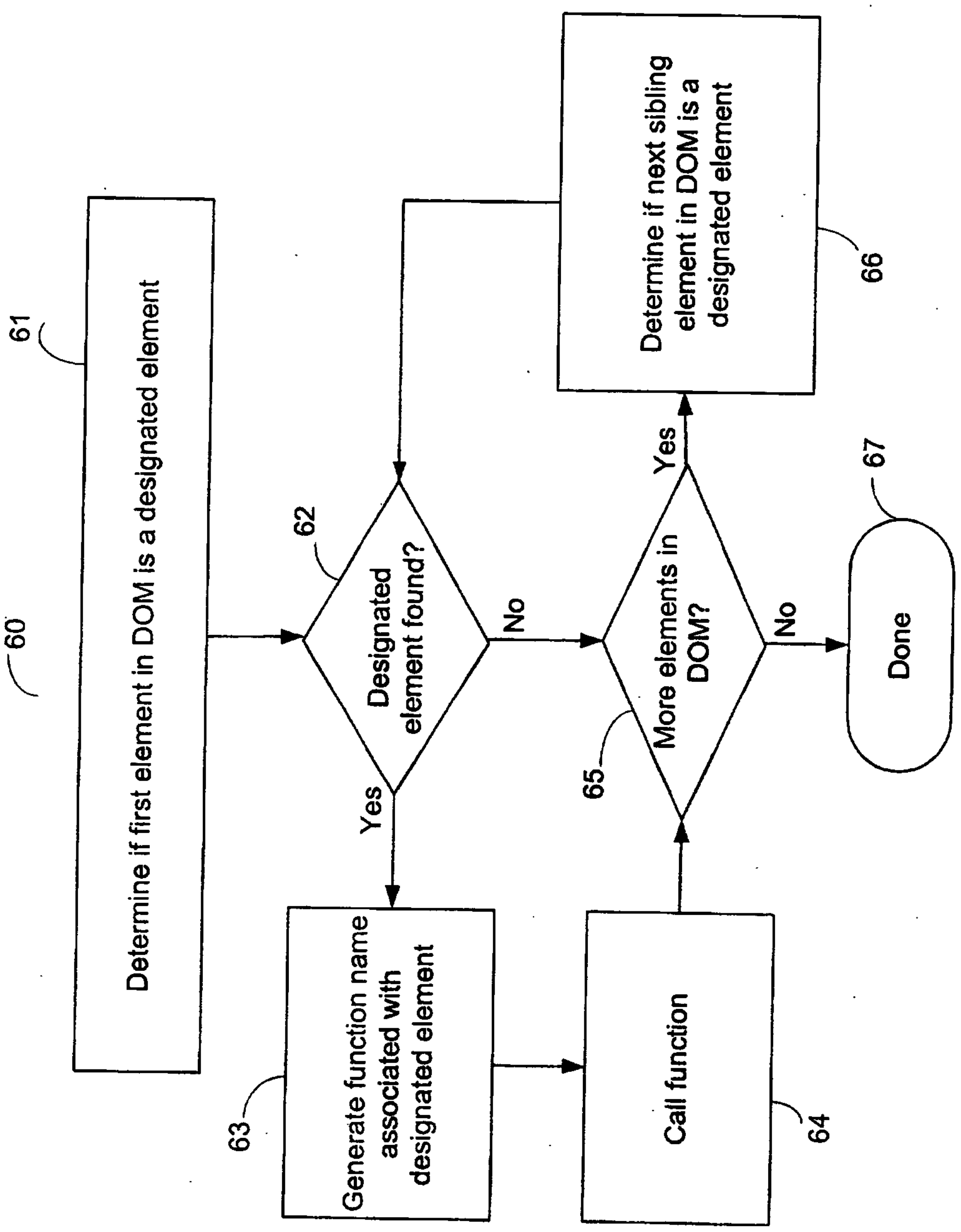
60
61
Determine if first element in DOM is a designated element
62
Designated element found?
Yes
No
63
Generate function name associated with designated element
64
Call function
65
More elements in DOM?
Yes
No
66
Determine if next sibling element in DOM is a designated element
67
Done

Figure 6

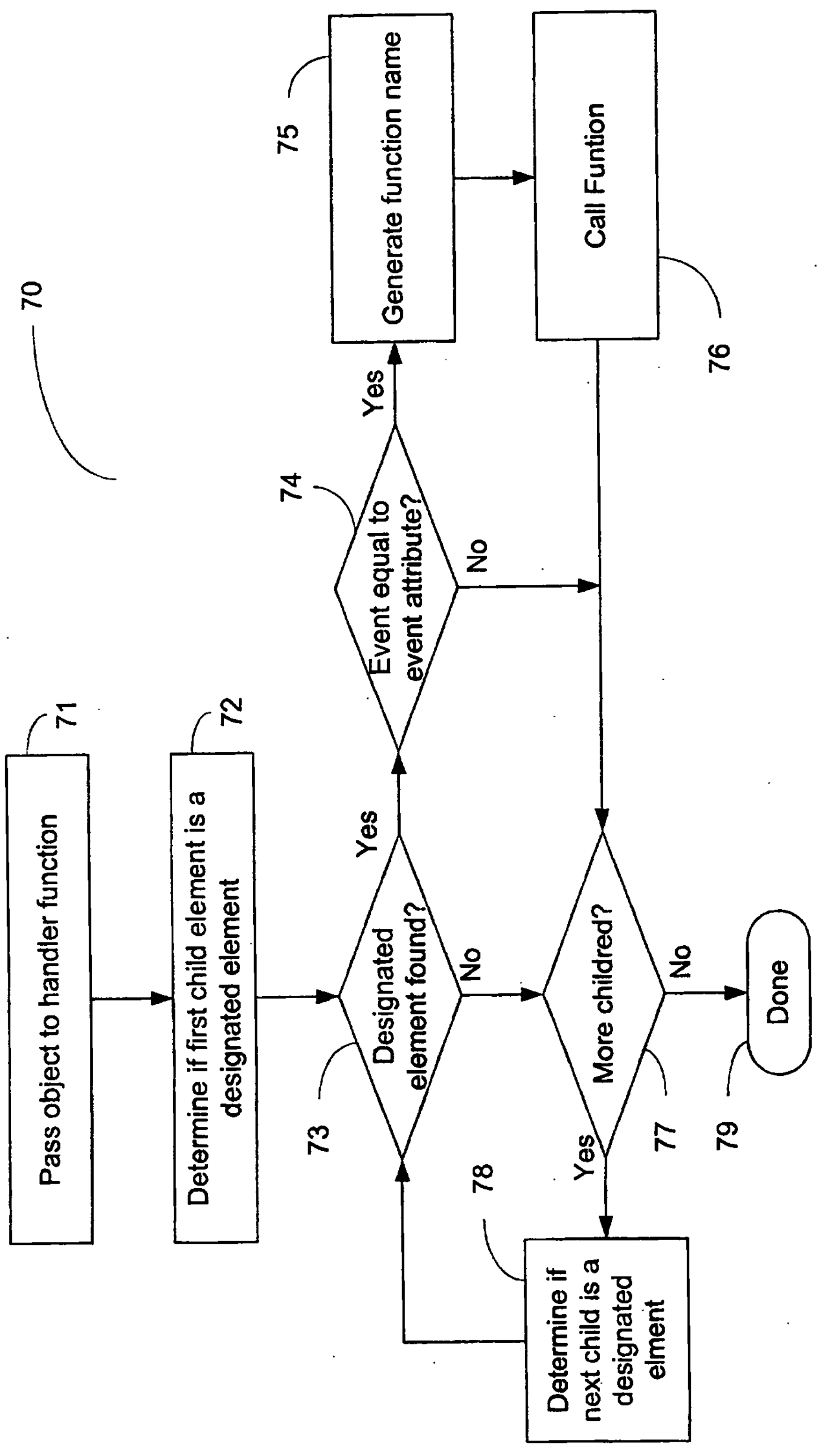
70
Pass object to handler function
71
Determine if first child element is a designated element
72
Designated element found?
73
Yes
No
Event equal to event attribute?
74
Yes
No
Generate function name
75
Call Funtion
76
More childred?
77
Yes
No
Determine if next child is a designated elment
78
Done
79

Figure 7

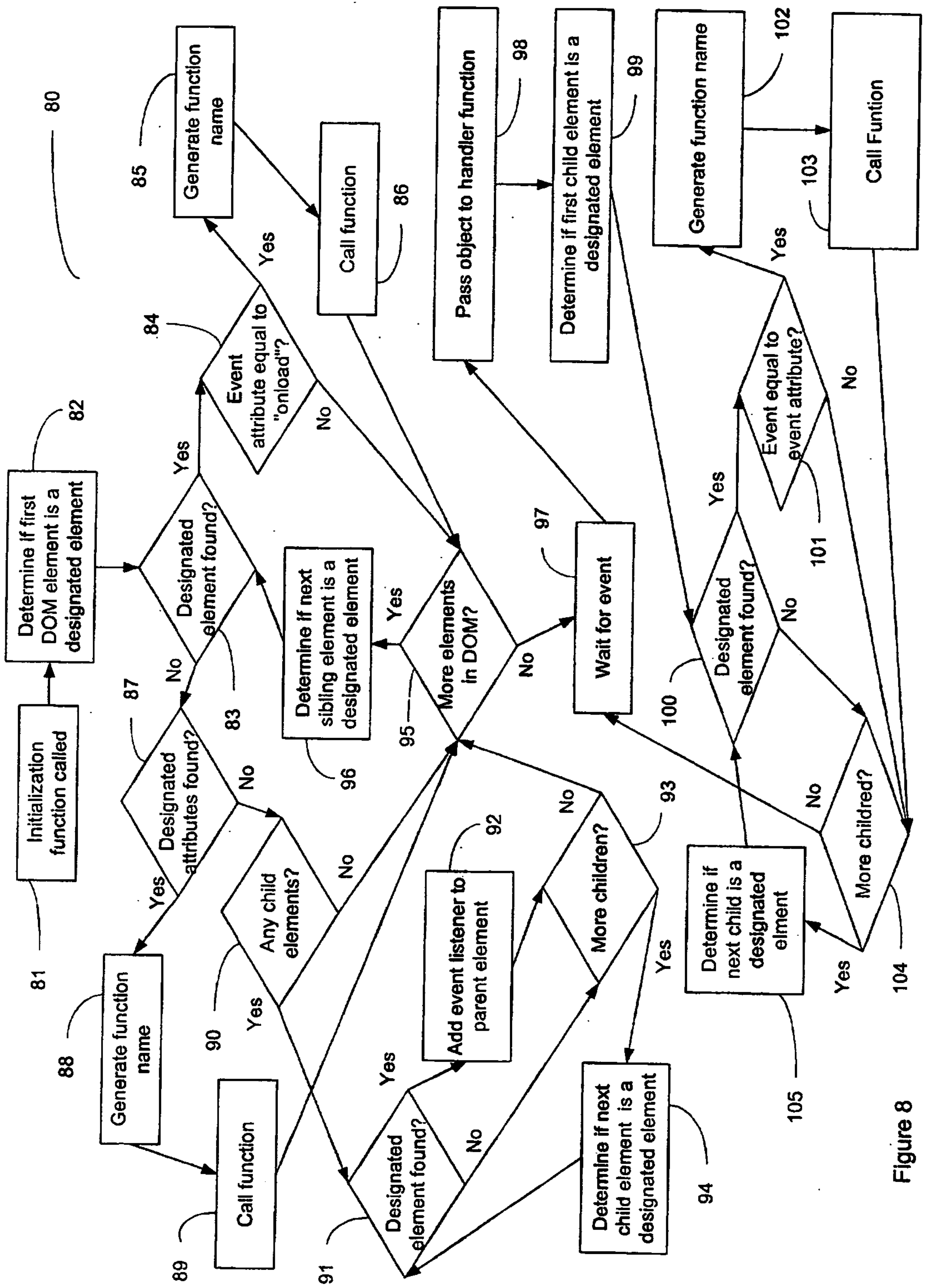
80
Initialization function called
81
Determine if first DOM element is a designated element
82
Designated element found?
83
Yes
No
Event attribute equal to "onload"?
84
Yes
No
Generate function name
85
Call function
86
Designated attributes found?
87
Yes
No
Generate function name
88
Call function
89
Any child elements?
90
Yes
No
Designated element found?
91
Yes
No
Add event listener to parent element
92
More children?
93
Yes
No
Determine if next child element is a designated element
94
More elements in DOM?
95
Yes
No
Determine if next sibling element is a designated element
96
Wait for event
97
Pass object to handler function
98
Determine if first child element is a designated element
99
Designated element found?
100
Yes
No
Event equal to event attribute?
101
Yes
No
Generate function name
102
Call Funtion
103
More childred?
104
Yes
No
Determine if next child is a designated elment
105

Figure 8

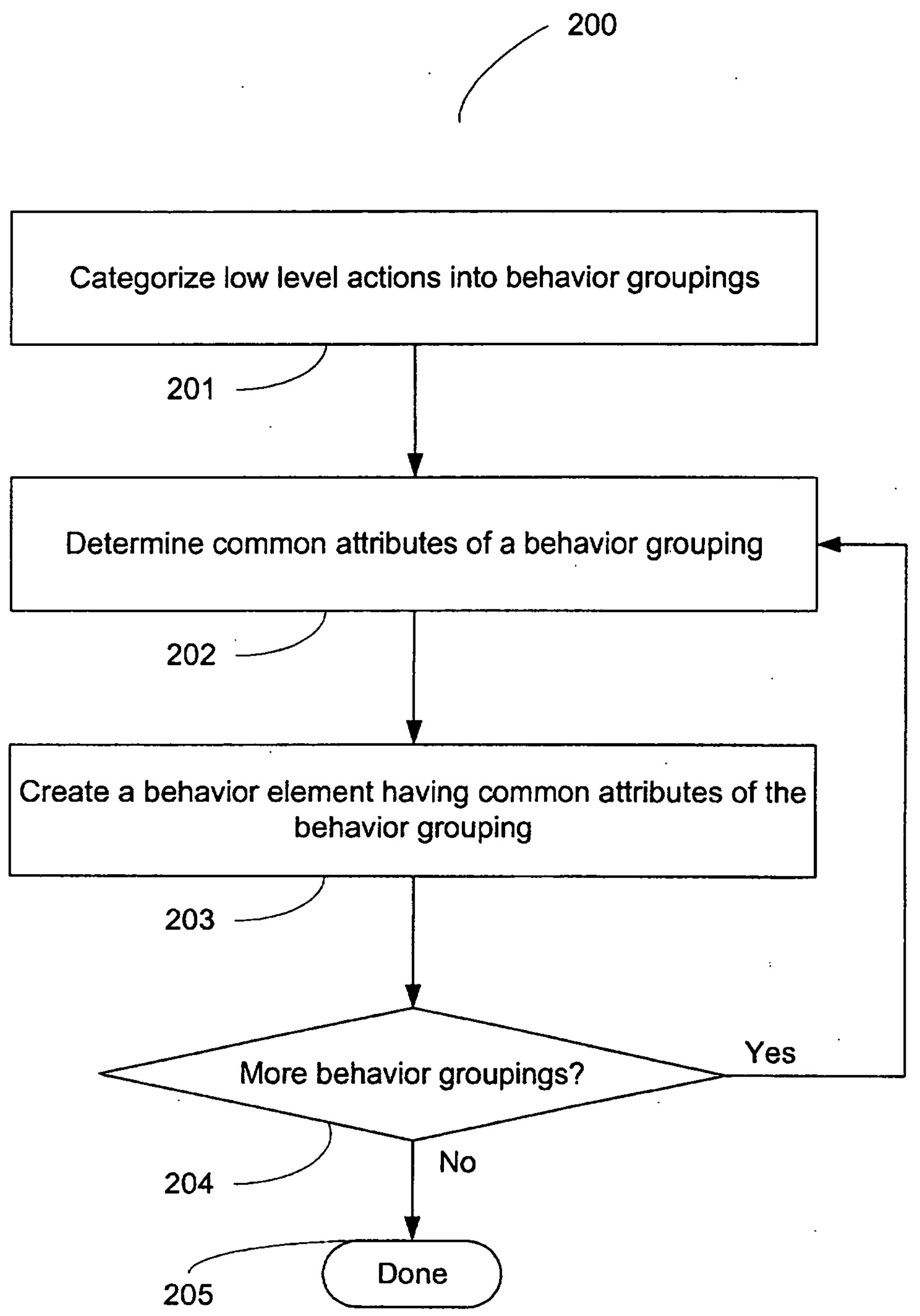
200
Categorize low level actions into behavior groupings
201
Determine common attributes of a behavior grouping
202
Create a behavior element having common attributes of the behavior grouping
203
More behavior groupings?
Yes
No
204
Done
205

Figure 9

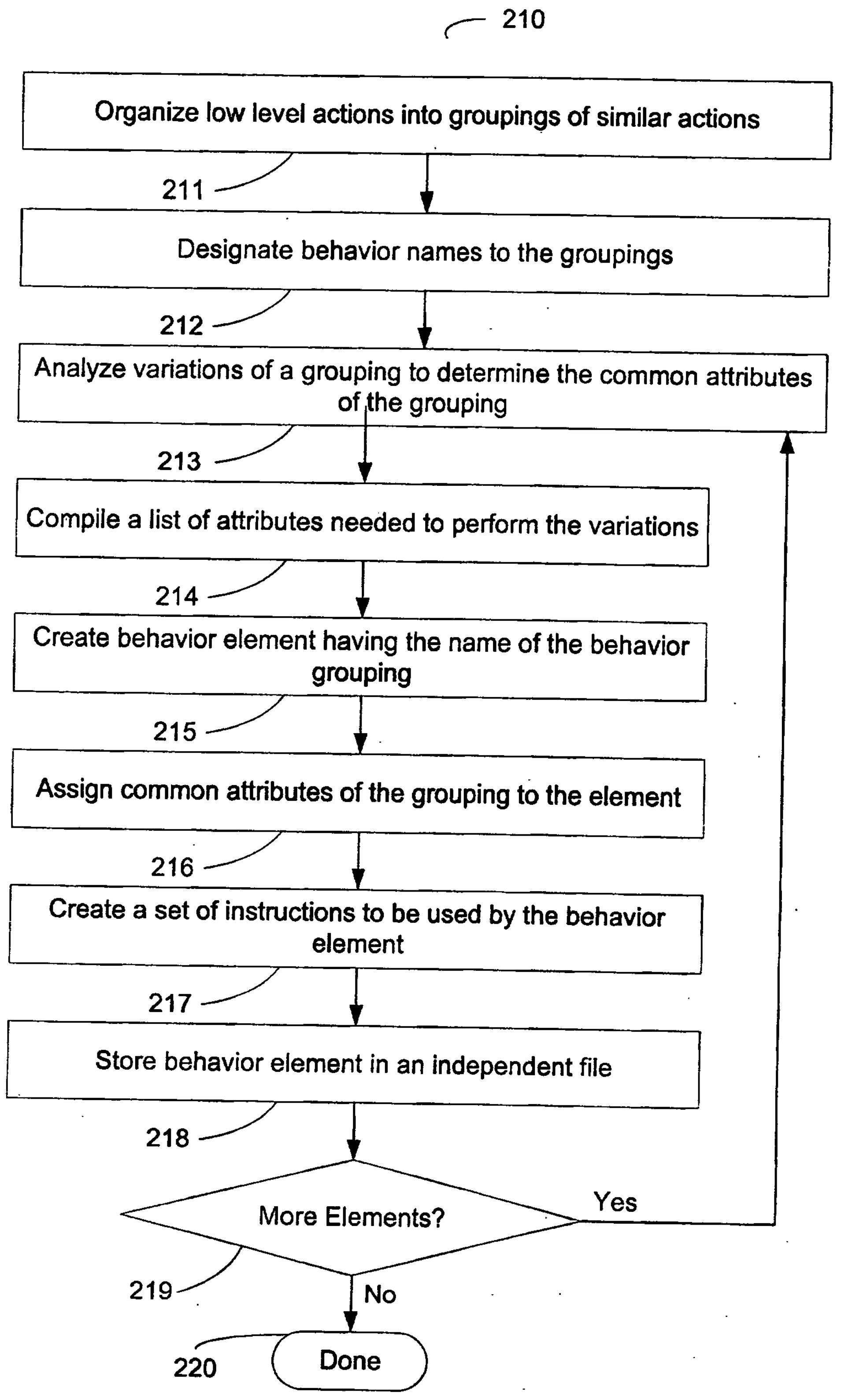
210
Organize low level actions into groupings of similar actions
211
Designate behavior names to the groupings
212
Analyze variations of a grouping to determine the common attributes of the grouping
213
Compile a list of attributes needed to perform the variations
214
Create behavior element having the name of the behavior grouping
215
Assign common attributes of the grouping to the element
216
Create a set of instructions to be used by the behavior element
217
Store behavior element in an independent file
218
More Elements?
Yes
219
No
220
Done

Figure 10